175 PTAS

58

mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR



Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen V - Fascículo 58

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-007-4 (tomo 5)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 208502
Impreso en España - Printed in Spain - Febrero 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

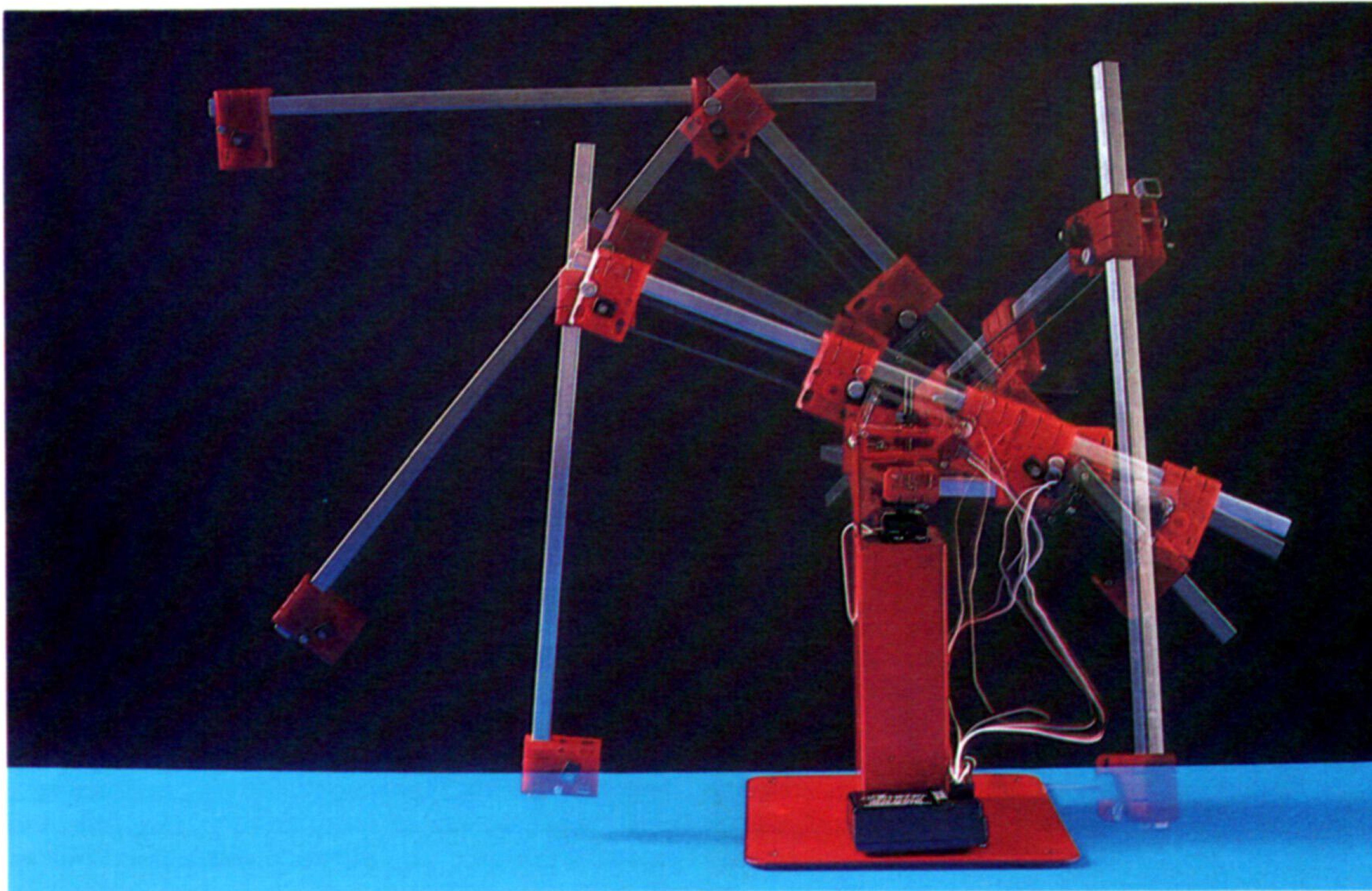
No se efectúan envíos contra reembolso.

Editorial Delta, S.A.



Brazos y manos

Ahora consideraremos un aspecto básico del diseño de un robot: el control y movimiento de las extremidades superiores



Ian McKinnell

Juntura doble
Durante la próxima década, el robot arquetípico será el brazo simple equipado con diversas "manos" para uso industrial, doméstico y para aficionados. Son muy pocas las aplicaciones que realmente requieren la máquina pensante autónoma y autopropulsada tan corriente en la ciencia-ficción, pero una uña semiinteligente programable es un dispositivo tan significativo como lo fueran en su tiempo el arado o el telescopio

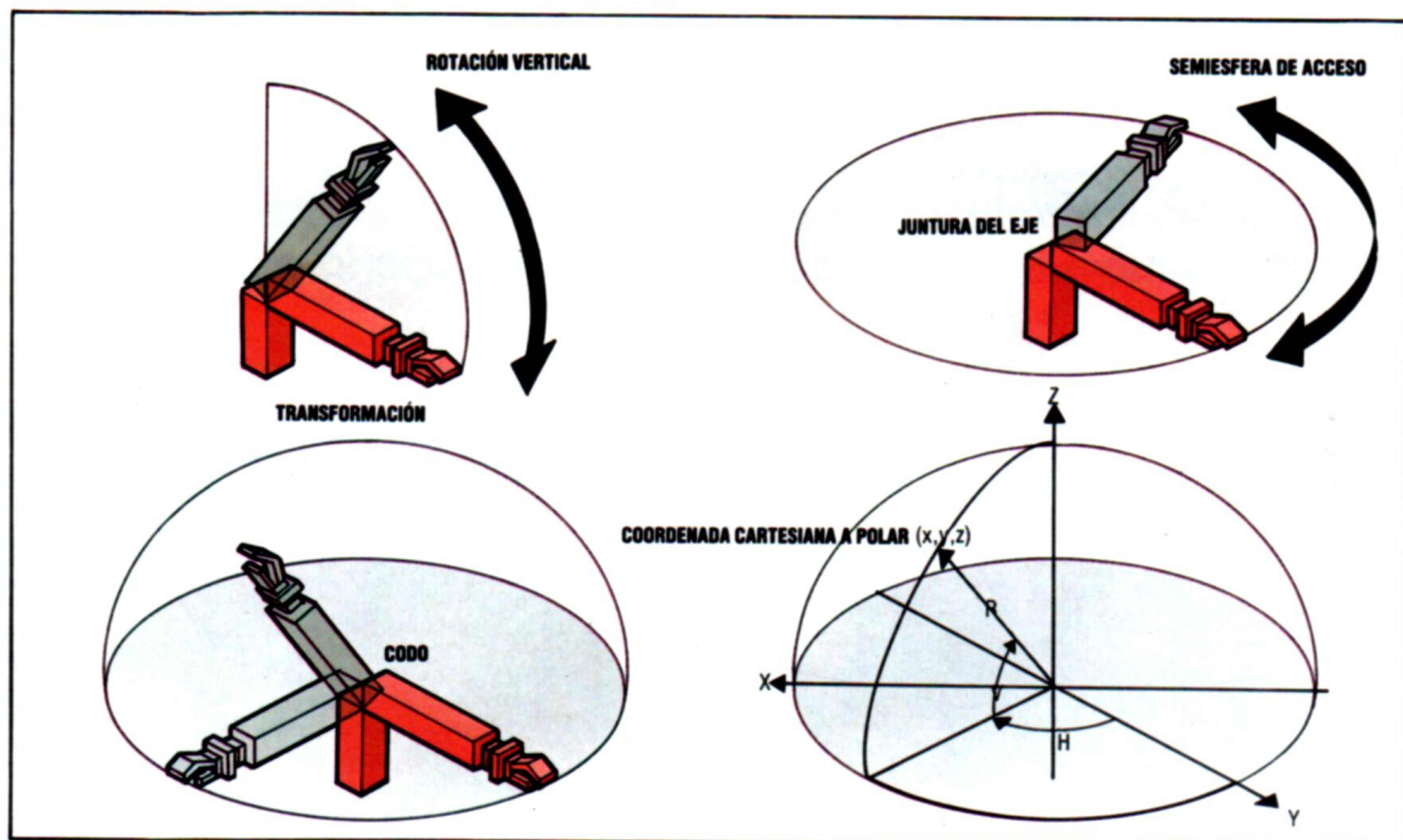
La eficacia de un robot depende en gran medida de la precisión con la cual pueda manipular objetos. Muchos robots se utilizan fundamentalmente para operaciones de "asir y colocar": trasladar componentes, en una planta industrial, de una cinta transportadora a otra, por ejemplo. Por tanto, el diseño del brazo-robot reviste la máxima importancia.

En general, son tres las exigencias que se deben considerar. Se debe desarrollar un sistema que describa la posición del brazo en cualquier momento dado; éste debe tener un "esqueleto" y debe haber un sistema "muscular" que accione el brazo y permita controlarlo. Las diferentes formas en las que interactúan estos tres elementos básicos suelen conformar la apariencia global de los brazos-robot. Sin embargo, en líneas generales se podría hacer una clasificación de distintos tipos de brazo atendiendo a los procedimientos espaciales utilizados para describir la posición del brazo en un momento dado.

En nuestro análisis del movimiento del robot (véase p. 1101), describimos el sistema de coordenadas cartesianas. Empleando este método, la posición del robot sobre el suelo se especificaba en virtud de dos ejes (x e y) en ángulo recto el uno del otro. A un brazo-robot se puede aplicar el mismo principio; pero, puesto que un brazo se puede mover libremente en tres dimensiones, es necesario que agreguemos otra variable (z) para describir la posición vertical del mismo. Utilizando estas coordenadas x, y y z podemos describir la posición del brazo en cualquier punto del espacio (siendo "espacio" simplemente la forma matemática de describir una superficie abierta).

Se puede construir un brazo-robot que se mueva exactamente a lo largo de estas tres coordenadas: lo que resulte se parecerá a una grúa de caballete elevada que se puede mover arriba/abajo, delante/atrás y de lado a lado (o en una combinación de las tres direcciones). Brazos como éste son muy apropiados para tareas en las que el trabajo se realiza en un área fija. Por ejemplo, al robot se lo puede colocar frente a una mesa de trabajo en la cual lleve a cabo todas sus tareas y, en este caso, un brazo cartesiano es más que adecuado. Pero este procedimiento tiene sus desventajas. Por ejemplo, los brazos de esta clase requieren una estructura especial del entorno, que los incapacita para realizar aplicaciones alejadas de la mesa de trabajo.

Otro método para describir la posición de un brazo emplea coordenadas cilíndricas. Para formarse una idea de cómo funcionan éstas, imagínese un bote de hojalata vacío; es fácil comprender que cualquier posición dentro del bote se puede describir especificando su distancia desde el centro del bote (utilizando una variable distancia, r); a qué distancia alrededor del bote está respecto a un punto fijo (empleando una variable angular, θ); y a qué distancia del lado del bote está (utilizando otra variable de distancia, z). De modo que recurriendo a las coordenadas cilíndricas sería fácil desarrollar un sistema que permitiera asir un objeto situado en una posición especificada dentro del bote.



Rotaciones robóticas
El brazo más simple, compuesto por una uña y una juntura de codo de dos ejes, es capaz de un posicionamiento de precisión en un volumen de espacio muy grande, tal como reflejan estas ilustraciones.
El codo gira sobre un gozne, permitiendo el movimiento vertical semicircular, y gira sobre un eje, lo que permite el movimiento circular horizontal. El brazo se desplaza hasta cualquier punto de la semiesfera de acceso mediante rotaciones en el eje y el gozne. Éstas se pueden obtener por trigonometría desde las coordenadas cartesianas (*x, y* y *z*) del punto tal como se indica: H, la rotación del eje, es igual a ARCTAN(*x/y*), mientras que V, el ángulo del gozne, es ARCSIN(*z*/R). El brazo se programa con las posiciones cartesianas de los objetos; transforma cada grupo de coordenadas en dos rotaciones que envía a sus servomotores, produciendo el movimiento

Los brazos que utilizan coordenadas esféricas llevan este proceso un paso más allá al especificar una posición en términos de dos ángulos y una distancia. En este caso, "distancia" equivale a la longitud del brazo, y los dos ángulos son la cantidad en función de la cual rota la base y el ángulo de elevación del brazo. Los brazos de este tipo se parecen mucho a una torreta de cañón, en la que se pueda variar la longitud del tubo del mismo. Las coordenadas esféricas se describen, por lo general, como r, θ y φ.

Para el ingeniero en robots es bastante sencillo diseñar un brazo que se pueda mover hacia dentro y hacia fuera telescópicamente, accionado, por ejemplo, mediante energía hidráulica.

El último y más corriente procedimiento para describir la posición de un brazo recurre a la utilización de otro tipo específico de coordenadas. Éste es un sistema diseñado especialmente para controlar brazos-robot imitando el accionar del brazo humano. Como antes, se necesitan tres variables para especificar la posición del brazo; esta vez son todas ángulos y se podrían describir como coordenadas θ, φ y γ. θ (*theta*) corresponde al ángulo a través del cual rota la base; φ (*phi*) representa el ángulo de elevación del brazo; y γ (*gamma*) describe el ángulo de una segunda juntura del brazo.

El sistema de coordenadas elegido determinará el tipo de "esqueleto" que requiera un robot. Todo lo que se necesita ahora es un poco de "músculo" para darle energía al movimiento del brazo. En general, son tres los tipos de músculos que se utilizan para los robots: eléctricos, hidráulicos y neumáticos. Analicémoslos separadamente.

Ya hemos hablado de la energía eléctrica en relación al movimiento del robot. Los mismos motores eléctricos paso a paso se podrían emplear para los brazos-robot. Por ejemplo, se pueden utilizar directamente, teniendo un potente motor en cada juntura del brazo y dejando que éste rote una pequeña cantidad para cada movimiento de juntura, o indirectamente, mediante engranajes, poleas o palancas.

Sin embargo, un sistema mejor implicaría hacer que los "músculos" del robot trabajaran de forma muy similar a los nuestros: estirándose y contrayéndose de modo que actúen directamente sobre el esqueleto del brazo. Esto se consigue disponiendo una serie de pistones para actuar sobre cada juntura del brazo. Estos pistones pueden ser hidráulicos (utilizando líquido) o neumáticos (empleando aire). Para los robots industriales se prefiere la energía hidráulica, porque puede proporcionar una presión mucho mayor (confiriéndole más fuerza al brazo) y porque el líquido no se dilata ni se contrae en la misma medida que el aire.

Esto significa que cuando un pistón se mueve a lo largo de un cilindro mediante presión hidráulica, no sufre efectos de "rebote", sino que se detiene exactamente en el punto deseado. El aire, por el contrario, no permite un posicionamiento de tanta precisión. Independientemente del sistema que se aplique, se pueden utilizar pistones de actuación simple o doble para producir movimiento en el brazo. Este tipo de fuerza motriz se denomina *de acción lineal*.

Aún es posible otro refinamiento. En vez de utilizar pistones que se muevan hacia atrás y hacia delante y luego traducir este movimiento en una rotación de la juntura, se puede recurrir a un *actuador rotatorio*. Éste produce una rotación directa en las junturas por medio de presión sobre un aspa dentro de una envoltura circular. Se trata de un proceso similar al empleo de un motor eléctrico paso a paso, pero la presión hidráulica significa que se puede ejercer muchísima más fuerza. La presión neumática no es apropiada para estas aplicaciones.

Una vez que se ha tomado una decisión sobre la mecánica del brazo-robot, ya sólo hace falta una "mano" (o *efector final*), de modo que, una vez posicionado correctamente el brazo, éste pueda hacer algo. Aquí resulta de ayuda pensar en la forma en que trabaja la mano humana. Consideremos la muñeca: si estuviera recubierta de yeso de manera que no se pudiera mover, la mayoría de las tareas resultarían mucho más difíciles. Cuando se pulsa un teclado, por ejemplo, las muñecas permiten que las manos se muevan hacia arriba y hacia abajo mien-

tras se golpean las teclas; esto se conoce como *cabezada* y sin ella al mecanografiar se tendría que mover todo el antebrazo hacia arriba y hacia abajo.

Las muñecas también se mueven hacia uno y otro lado a medida que se van pulsando las distintas teclas; esto se denomina *guiñada*, y su ausencia supondría el movimiento del codo. Cuando se termina de teclear, se pueden girar las muñecas de modo que las manos descansen con el pulgar hacia arriba junto al teclado. Esto se conoce como *tonel* y, de no disponer del movimiento de muñeca, sería necesario un complicado juego de movimientos de hombro.

En un plano ideal, estos tres distintos movimientos se deberían incorporar en la muñeca del robot. Cada uno de los movimientos (cabezada, guiñada y tonel) puede actuar en dos direcciones (arriba/abajo, izquierda/derecha, en sentido horario/antihorario) y cada combinación de movimiento y dirección se conoce como *grado de libertad*. En consecuencia, se puede decir que un robot que incorpore cabezada, guiñada y tonel posee seis grados de libertad. Los robots se construyen con menos grados de libertad, generalmente cuatro o cinco, pero cada reducción en el movimiento de muñeca queda compensada con un aumento de los movimientos que deben efectuar los otros miembros, más largos, del brazo.

La mano del robot

Ahora debemos considerar el diseño de la mano propiamente dicha. La configuración ideal sería una mano similar a la humana dispuesta en el extremo de un brazo semejante al del hombre; ya es posible hallar manos-robot que respondan a esta definición. La forma más común de mano-robot es una uña de tres dedos (compuesta por dos dedos más el "pulgar" opuesto) que permite que el robot coja objetos de una forma muy parecida a como lo haría una mano humana.

La energía que se utiliza para activar la mano puede ser de cualquiera de los tres tipos ya mencionados, y dependerá de la tarea que haya de llevar a cabo el robot. Si la mano debe mover objetos grandes que pesen alrededor de un centenar de hilos, probablemente será necesaria la hidráulica. Pero para muchas aplicaciones bastará la energía eléctrica o neumática, porque la mano sólo necesitará asir el objeto y soltarlo cuando se desee: si el brazo y la muñeca han posicionado correctamente la mano, ello no requerirá ninguna precisión; un simple movimiento de apertura y cierre será suficiente.

En muchos casos, no obstante, el brazo-robot no estará dotado de mano. Ya hemos empleado los términos "efector final" para describir una mano, pero estas palabras pueden igualmente aludir a otras muchas cosas. Un robot que se emplee para soldar no requiere mano en absoluto: se puede instalar,una pistola soldadora directamente en la muñeca. En realidad, algunos robots son capaces de elegir el efector final correcto para la tarea que estén llevando a cabo; pueden descartar un efector final (un destornillador, supongamos) e insertar otro (una pistola spray, p. ej.) en un conector estándar en la muñeca. Puede que ésta no sea una acción de índole particularmente humana, pero sirve para hacer que los robots sean en extremo adaptables.

Muñeca del robot
Si el sencillo brazo de dos ejes que vemos en la ilustración se equipa con una muñeca flexible, entonces su espacio accesible aumenta enormemente; la complejidad de las transformaciones de coordenadas se incrementa con la articulación de la muñeca, pero las operaciones matemáticas llegan a parecer casi irrelevantes en comparación con los problemas de ingeniería que implica el coordinar cualidades como poco peso, fortaleza, precisión y flexibilidad. El verdadero problema de software es "decidir" cuál, entre las numerosas orientaciones posibles de uñas, se presta mejor para los objetos a asir

Cabezada y guiñada
La capacidad de la muñeca humana de hacer cabezadas y guiñadas (movimiento arriba/abajo e izquierda/derecha) está limitada precisamente por las restricciones que afectan a los diseñadores de robots, es decir, el problema de conciliar junturas fuertes y compactas con la necesidad de flexibilidad y poco peso. Estas junturas no agregan gran cosa al espacio accesible, pero sí incrementan enormemente la capacidad del brazo para manipular objetos

Tonel y extensión
Estas junturas son las más fáciles de construir y contribuyen mucho más al espacio accesible del brazo que las junturas de cabezada y guiñada. Los brazos dotados sólo de capacidades para tonel y extensión son apropiados para la mayor parte de las aplicaciones industriales

Nombres de variables

Iniciamos una serie en la que desarrollaremos una gama de programas de utilidad para los ordenadores más populares

La gama de utilidades que emplean los micros personales varía considerablemente: algunos ordenadores sólo poseen un editor sencillo, como el Sinclair Spectrum, mientras que otras máquinas configuran facilidades más amplias. El BBC Micro, por ejemplo, incluye las instrucciones TRACE y RENUM (trazar y renumerar): la primera hace que a medida que se ejecutan las instrucciones BASIC se vaya visualizando su número de línea, y la segunda renumera automáticamente las líneas de un programa en BASIC. Estas dos facilidades son de enorme utilidad en el desarrollo y la depuración de programas. Pero, sea lo que sea lo que se proporcione con su máquina, siempre es conveniente disponer de utilidades adicionales, y la gama de programas disponibles a nivel comercial entre los cuales se puede escoger es muy amplia.

Los programas de utilidad por lo general se escriben en lenguaje assembly, debido en parte a la velocidad del código máquina y también a que no es fácil que un programa en BASIC se modifique a sí mismo sin que se "cuelgue" el sistema. No obstante, empezaremos analizando algunas utilidades sencillas que se pueden escribir en BASIC. De esta manera, podemos concentrar nuestra atención en lo que un programa de utilidad tiene que hacer, sin que sea preciso entrar a considerar otros detalles que entrañarían complicaciones, tales como el papel del sistema operativo del ordenador y del intérprete de BASIC.



Buscando Hi y Lo
En el BBC Micro y el Spectrum, una línea de programa empieza con tres o cuatro bytes dedicados al número de línea y a la longitud de la misma. A esto le sigue el texto de BASIC por elementos. Cuando el programa encuentra la línea, registra el número de ésta y calcula la dirección de comienzo de la siguiente a partir de la longitud de la línea en curso. Luego "desliza" la plantilla de búsqueda a lo largo del texto de programa hasta que encuentra final-de-línea, final-de-programa, un distintivo REM o una pareja que coincide con la buscada

Si bien es difícil que un programa en BASIC se altere a sí mismo, no existe ningún problema en crear un programa en este lenguaje que inspeccione a otro programa BASIC. El programa de utilidad que ofrecemos aquí, en versiones para el Spectrum y el BBC Micro, busca a través de un programa BASIC el nombre de una variable o función, e imprime los números de las líneas en que se halla el nombre.

Ambos programas comienzan por buscar en qué lugar de la memoria del ordenador empieza el texto del programa. Luego revisan el programa línea por línea, saltándose aquellas secciones que no pueden incluir un nombre y extrayendo todos éstos. El paso final implica comparar cada uno de los nombres extraídos con el nombre cuya búsqueda se solicitó al programa.

Cuando el programa comienza la búsqueda en una nueva línea del texto en BASIC, primero apunta el número de línea, que en ambos casos se almacena en dos bytes, y la longitud de la línea (el número de bytes que ocupa). En el BBC Micro, la longitud de la línea está contenida en un byte, y es la cantidad total de bytes de la línea desde el número de línea hasta el indicador de final de línea (código ASCII 13). En el Spectrum, la longitud de la línea se almacena en dos bytes y representa la cantidad de bytes desde el carácter que sigue a los bytes de longitud hasta el indicador de final de línea (sin incluir, por lo tanto, en su total, el número de línea y los bytes que indican su longitud).

En ambas versiones del programa hacemos caso omiso de las sentencias REM y todo cuanto esté encerrado entre comillas, puesto que normalmente en estas series de caracteres no tendremos ninguna variable de programa. El BBC Micro permite la inclusión de números hexadecimales en un programa, precedidos por el carácter &. Necesitamos asegurarnos de que nuestro programa no confunda estos números hexadecimales tomándolos por nombres de variables y, por consiguiente, tenemos que hacer que el programa se salte todas las series que vayan precedidas por un &. Por ejemplo, no deseamos que el programa tome al número hexa A0 de &A0 como el nombre de la variable "A0".

En el Spectrum, los números se almacenan en un programa como los caracteres ASCII correspondientes a los dígitos del número, seguidos por el byte 14 del código ASCII, y luego cinco bytes que contienen el equivalente binario del número. Nuestro programa ha de ser capaz de saltarse el código del número y su equivalente binario de cinco bytes.

Habiendo verificado estas condiciones el programa prosigue inspeccionando la línea en curso en busca de cualquier nombre que contenga. En ambos programas, un nombre, por definición, empieza con una letra, seguida de otra letra o un dígito. La versión para el BBC Micro admite variables enteras (distinguidas por un carácter % después del nombre) y el carácter de subrayado, y las dos versiones admiten variables en serie (*strings*), que van seguidas de un carácter $.

El nombre de una matriz, una función o (en el BBC Micro) un procedimiento irá seguido por un paréntesis abierto:(. En realidad, éste no forma parte del nombre, pero los programas los utilizan para distinguir a éstos de las variables simples.

En el programa para el Spectrum se plantean otras complicaciones. En particular, el hecho de que el BASIC Spectrum no realiza ninguna distinción entre caracteres en mayúscula y caracteres en minúscula del nombre de una variable. Por consiguiente, JUAN, Juan o JUan se tratan todos como el mismo nombre de variable. El programa del Spectrum, en consecuencia, convierte todas las letras a mayúsculas antes de iniciar su comparación. Asi-



mismo, el Spectrum permitirá incluir espacios en los nombres de variables, pero éstos pueden plantear problemas. Nuestro consejo es abstenerse de utilizarlos.

El BASIC Spectrum no realiza la distinción estricta que efectúan muchas otras versiones del BASIC entre variables en serie y matrices en serie; en realidad, una variable en serie del Spectrum es más bien una matriz de caracteres. Dado que podemos tener, por ejemplo, T$ y T$(i), en referencia a una serie y parte de la misma serie, el programa no intenta distinguir entre variables en serie simples y matrices en serie. Ésta no es, sin embargo, una limitación, puesto que el BASIC Spectrum no permite tener una variable en serie y una matriz que posean el mismo nombre.

Para emplear nuestro programa de utilidad, primero éntrelo y guárdelo, y después mézclelo con el programa en el que desea realizar la búsqueda, valiéndose de la instrucción MERGE en el Spectrum o siguiendo el procedimiento que se describe en el capítulo 37 de la guía para el usuario en el caso del BBC Micro. Invoque la rutina de búsqueda con RUN 9000 (Spectrum) o GOTO 30000 (BBC Micro), y digite el nombre que desea hallar cuando así se le solicite. Si desea un nombre de matriz, al final del nombre agregue un paréntesis abierto.

Por último, a modo de ejercicio, quizá desee utilizar los principios aquí descritos para escribir un programa que le diga cuáles son las líneas de un programa que contienen una llamada a una subrutina específica. En el próximo capítulo ofreceremos un listado completo para el Commodore 64.

Spectrum

```
9000 INPUT "Nombre a buscar? ";LINE T$
9010 FORi=1 T' LEN(T$)
9020 IF T$(i)>="a" AND T$(i)<="z" THEN LET
     T$(i)=CHR$(CODE(T$(i))-32)
9030 NEXT i
9040 LET DistintivoparaREM=234
9050 LET Comilla=34
9060 LET Nuevalinea=13
9070 LET Subrayado=95
9080 LET Numero=14
9090 LET PROG=23635
9100 LET Apuntadortexto=PEEK (PROG)+256*PEEK
     (PROG+1)
9110 LET Numlinea=256*PEEK (Apuntadortexto)+PEEK
     (Apuntadortexto+1)
9120 IF Numlinea>=9000 THEN STOP
9130 LET Apuntadortexto=Apuntadortexto+2
9140 LET Longitudtexto=PEEK (Apuntadortexto)+256*PEEK
     (Apuntadortexto+1)
9150 LET Apuntadortexto=Apuntadortexto+2
9160 LET Siguientelinea=Apuntadortexto+Longitudtexto
9170 IF PEEK (Apuntadortexto)=Nuevalinea THEN LET
     Apuntadortexto=Apuntadortexto+1: GO TO 9110
9180 IF PEEK (Apuntadortexto)<>DistintivoparaREM THEN GO
     TO 9220
9190 REM Saltar linea de REM
9200 LET Apuntadortexto=Siguientelinea
9210 GO TO 9110
9220 IF PEEK (Apuntadortexto)<>Comilla THEN GO TO 9280
9230 REM Saltar todo lo que haya entre comillas, pero detenerse
     al final de la linea en caso de comilla sin pareja
9240 LET Apuntadortexto=Apuntadortexto+1
9250 IF PEEK (Apuntadortexto)=Nuevalinea THEN LET
     Apuntadortexto=Apuntadortexto+1: GO TO 9110
9260 IF PEEK (Apuntadortexto)<>Comilla THEN GO TO 9240
9270 LET Apuntadortexto=Apuntadortexto+1
9275 GO TO 9170
9280 IF PEEK (Apuntadortexto)<>Numero THEN GO TO 9320
9290 REM Saltarse numero binario 5 bytes
9300 LET Apuntadortexto=Apuntadortexto+6
9305 GO TO 9170
9310 REM El primer caracter de un nombre debe ser una letra en
     mayuscula o minuscula
9320 IF PEEK (Apuntadortexto)>=CODE ("A") AND PEEK
     (Apuntadortexto)<=CODE ("Z") THEN LET
     c$=CHR$(PEEK(Apuntadortexto)):GO TO 9370
9330 REM Utilizar mayusculas en vez de minusculas
9340 IF PEEK (Apuntadortexto)>=CODE("a) AND PEEK
     (Apuntadortexto)<=CODE("z") THEN LET c$=CHR$
     (PEEK (Apuntadortexto)-32): GO TO 9370
9350 LET Apuntadortexto=Apuntadortexto+1
9360 GO TO 9170
9370 LET n$=""
9380 LET n$=n$+c$
9390 LET Apuntadortexto=Apuntadortexto+1
9400 REM Letra,digito o subrayado despues del primer caracter
     del nombre
9410 IF PEEK (Apuntadortexto)>=CODE("A") AND PEEK
     (Apuntadortexto)<=CODE ("Z") THEN LET c$CHR$
     (PEEK (Apuntadortexto)): GO TO 9380
9420 REM Utilizar mayusculas en lugar de minusculas
9430 IF PEEK (Apuntadortexto)>=CODE("a") AND PEEK
     (Apuntadortexto)<=CODE("z") THEN LET c$=CHR$
     (PEEK(Apuntadortexto)-32): GO TO 9380
9440 IF PEEK (Apuntadortexto)>=CODE("0") AND PEEK
     (Apuntadortexto)<=CODE("9") THEN LET c$=CH$
     (PEEK(Apuntadortexto)):GO TO 9380
9450 IF PEEK (Apuntadortexto)=Subrayado THEN GO TO 9380
9460 REM Final con $ siguiendo a variable en serie
9470 IF PEEK (Apuntadortexto)=CODE ("$") THEN LET
     n$=n$+"$": LET Apuntadortexto=Apuntadortexto+1:
     GO TO 9500
9480 REM Si matriz o función (
9490 IF PEEK (Apuntadortexto)=CODE("(") THEN LET
     n$=n$+CHR$ (PEEK (Apuntadortexto)): LET
     Apuntadortexto=Apuntadortexto+1
9500 IF n$=t$ THEN PRINT n$;"  EN LINEA  ";Numerolinea
9520 GO TO 9170
```

BBC Micro

```
30000 INPUT "Nombre a buscar";OBJETIVO$
30010 DistintivoparaREM=244
30020 Comilla=34
30030 Hexa=38
30040 Nuevalinea=13
30050 Subrayado=95
30060 Apuntadortexto=PAGE
30070 Apuntadortexto=Apuntadortexto+1
30080 Numerolinea=256*?Apuntadortexto+?(Apuntadortexto+1)
30090 IF Numerolinea>=30000 THEN END
30100 Apuntadortexto=Apuntadortexto+2
30110 Longitudlinea=?Apuntadortexto
30120 Finallinea=Apuntadortexto+Longitudlinea-3
30130 Apuntadortexto=Apuntadortexto+1
30140 IF ?Apuntadortexto=Nuevalinea THEN GOTO 30070
30150 IF ?Apuntadortexto<>DistintivoparaREM THEN GOTO
      30180
30160 REM Saltar linea REM
30170 Apuntadortexto=Finallinea:GOTO 30070
30180 IF ?Apuntadortexto<>Comilla THEN GOTO 30240
30190 REM Saltar todo lo que haya entre comillas, pero
      detenerse al final de linea en caso de comilla sin pareja
30200 Apuntadortexto=Apuntadortexto+1
30210 IF ?Apuntadortexto=Nuevalinea THEN GOTO 30070
30220 IF ?Apuntadortexto<>Comillas THEN GOTO 30200
30230 Apuntadortexto=Apuntadortexto+1
30235 GOTO 30140
30240 IF ?Apuntadortexto<>Hexa THEN GOTO 30300
30250 REM Saltar numero hexa, para evitar confusión con
      nombres de variables
30260 Apuntadortexto=Apuntadortexto+1
30270 IF ?Apuntadortexto>=ASC("0") AND
      ?Apuntadortexto<=ASC("9") THEN GOTO 30260
30280 IF ?Apuntadortexto>=ASC("A") AND
      ?Apuntadortexto<=ASC("F") THEN GOTO 30260
30285 GOTO 30140
30290 REM El primer caracter de un nombre debe ser una letra
      en mayuscula o minuscula
30300 IF ?Apuntadortexto>=ASC("A") AND
      ?Apuntadortexto<=ASC("Z") THEN GOTO 30330
30310 IF ?Apuntadortexto>=ASC("a") AND
      ?Apuntadortexto<=ASC("z") THEN GOTO 30330
30320 GOTO 30130
30330 Nombre$=""
30340 Nombre$="Nombre$+CHR$(?Apuntadortexto)
30350 Apuntadortexto=Apuntadortexto+1
30360 REM Letra, digito o subrayado despues de primer
      caracter
30370 IF ?Apuntadortexto>=ASC("A") AND
      ?Apuntadortexto<=ASC("Z") THEN GOTO 30340
30380 IF ?Apuntadortexto>=ASC("a") AND
      ?Apuntadortexto<=ASC("z") THEN GOTO 30340
30390 IF ?Apuntadortexto>=ASC("0") AND
      ?Apuntadortexto<=ASC("9") THEN GOTO 30340
30400 IF ?Apuntadortexto=Subrayado THEN GOTO 30340
30410 REM Final con $ para variable en serie, % para variable
      entera
30420 IF ?Apuntadortexto=ASC("$") THEN
      Nombre$=Nombre$+CHR$(?Apuntadortexto):
      Apuntadortexto=Apuntadortexto+1: GOTO 30450
30430 IF ?Apuntadortexto=ASC("%") THEN
      Nombre$=Nombre$HR$(?Apuntadortexto):
      Apuntadortexto=Apuntadortexto+1
30440 REM (si matriz, procedimiento o funcion
30450 IF ?Apuntadortexto=ASC("(") THEN
      Nombre$=Nombre$+CHR$(?Apuntadortexto):
      Apuntadortexto=Apuntadortexto+1
30460 If Nombre$=OBJETIVO$ THEN PRINT Nombre$;"  EN
      LINEA  ";Numerolinea
30470 GOTO 30140
30480 END
```

Sprites en LOGO

Después de haber estudiado con cierto detalle la geometría de tortuga, avanzaremos en nuestro curso examinando el empleo de sprites

Utilizando LOGO, los sprites se comportan de forma similar a la tortuga, obedeciendo todas las instrucciones que cumple ésta. Sin embargo, a diferencia de la misma, nosotros podemos definir la forma de un sprite, si bien estas formas no rotan en la pantalla cuando cambia el encabezamiento del sprite como lo suele hacer una tortuga.

En LOGO Commodore, la tortuga se cuenta como el sprite número 0, y hay otros siete sprites (numerados del 1 al 7). Para empezar, el sprite 0 es el sprite "en curso" y obedece todas las instrucciones para sprites que se generan. Para hacer que el sprite 1 sea el sprite en curso, simplemente debe ejecutarse TELL 1. A partir de entonces, el sprite 1 obedecerá todas las instrucciones para sprites hasta que se especifique un sprite en curso diferente.

No obstante, después de ejecutar TELL 1, no se verá nada en la pantalla. Ello se debe a que todos los sprites, excepto la tortuga, empiezan como objetos "ocultos" y tienen sus "lápices" hacia arriba. Para poder ver el sprite 1 y su trayectoria, debe ejecutarse ST, y en la pantalla aparecerá un cuadrado impreciso. Experimente con esta cuadrícula de sprite utilizando las instrucciones de tortuga FD,BK,RT,LT,PD,PU,ST,HT, etc.

Si se desplaza el sprite 1 hasta la misma posición que el sprite 0 (la tortuga), notará que el sprite 1 parecerá que está detrás de la tortuga. En general, los sprites de número inferior se muestran "por delante" de los sprites de números superiores. Esto es muy útil para crear efectos tridimensionales.

En el disco de utilidades del LOGO Commodore hay un editor de sprites. Cárguelo digitando READ"SPRED. Para editar la forma del sprite primero conviértalo en el sprite en curso mediante TELL 1 y después digite EDSH. Entonces se visualizará la forma del sprite muy ampliada, y nos permitirá desplazar el cursor por la pantalla. Al pulsar la tecla del asterisco (*) se activará un pixel; si se pulsa la barra espaciadora se borrará.

Habiendo diseñado su sprite, pulse Control-C para definir la forma. Si el sprite no está visible, pruebe de ejecutar ST. Esta misma forma se le puede dar ahora también a otros sprites. SETSHAPE 1 (determinar forma) le proporcionará al sprite en curso la misma definición que al sprite 1. Después de haber definido un grupo de formas, puede guardar los sprites en un archivo con SAVESHAPES "NOMBREARCHIVO, y volver a leerlos con READSHAPES "NOMBREARCHIVO.

Hay un problema matemático muy conocido en el que se colocan cuatro insectos en las esquinas de un cuadrado. Todos parten a la misma velocidad y cada uno sigue al insecto que tiene a su derecha. El objetivo consiste en trazar sus recorridos. Aquí ofrecemos un programa en LOGO que implementa el problema utilizando sprites.



Los procedimientos que proporcionamos sitúan una copia del mismo sprite en cada esquina del cuadrado y luego los ponen en marcha siguiéndose los unos a los otros. La forma del insecto se define como el sprite 3 y a todos los otros se les da la misma forma utilizando SETSHAPE 3 en el procedimiento de posicionamiento.

El núcleo de nuestra solución está en el procedimiento SEGUIR. En éste, X e Y se establecen primero en las coordenadas *x* e *y* del sprite que se está siguiendo (:B) y luego el sprite que está haciendo el seguimiento (:A) tiene su encabezamiento orientado hacia ese punto. Para hacer esto, empleamos la primitiva TOWARDS (hacia). Ésta acepta dos parámetros de entrada, que representan las coordenadas del punto hacia el cual orientarse, y produce el encabezamiento del sprite en curso hacia ese punto.

Animación

Una interesante aplicación de los gráficos sprite es la creación de efectos de animación. Se define una serie de formas de sprite que representen el mismo objeto. Cada una de éstas es ligeramente diferente de la anterior, y cuando se ejecutan juntas crean un efecto de movimiento. El LOGO Commodore ofrece tres formas que crean la primaria figura de un hombre que corre. Los siguientes procedimientos preparan la pantalla y después ponen las tres formas en movimiento.

```
TO CORRER
   TELL 0
   DRAW
   PU
   BIGX BIGY
   SETH 90
   CORRIENDO 2
END
TO CORRIENDO :FORMA
   FD 5
   SETSHAPE :FORMA
   IF :FORMA=4 THEN MAKE "FORMA 1
   CORRIENDO :FORMA+1
END
```



Antes de ejecutar estos procedimientos, cargue el archivo SPRITES del disco de utilidades. Éste contiene unos cuantos procedimientos muy útiles, incluyendo BIGX y BIGY (grande), los cuales duplican el tamaño de un sprite. SMALLX y SMALLY (pequeño) son los procedimientos contrarios: se emplean para devolver un sprite a su tamaño original. Cargue los tres sprites digitando READSHAPES"RUNNER, y después ejecute los procedimientos.

En la página siguiente también definimos cuatro formas de sprite, que usaremos luego en un juego.



Complementos al LOGO

Ni el LOGO Spectrum ni el LOGO Apple poseen la facilidad de gráficos sprite. Los usuarios de Atari deben considerar estas diferencias:
1) Sólo hay cuatro sprites disponibles.
2) Para SETSHAPE utilizar SETSH.
3) El editor de sprites está incluido entre las primitivas. Pulsando la barra espaciadora se rellena un pixel vacío o se vacía uno lleno.

El dragón feroz

En el próximo capítulo publicaremos el juego del caballero y el dragón, que utiliza las facilidades de sprites del LOGO Commodore para explorar el algoritmo de persecución de la demostración de los insectos. Los sprites que vemos son formas que sugerimos para emplear en el juego

DRAGÓN

CABALLERO

CIUDAD 1

CIUDAD 2

Liz Dixo

Cuatro insectos

En esta demostración de geometría, cada insecto avanza directamente hacia la posición del insecto que tiene a su derecha. Este algoritmo produce una espiral hacia dentro; sus brazos describen la "curva de persecución" que tan familiar les resulta a los pilotos de combate y a los aficionados a los juegos recreativos

```
TO INSECTOS
   PREPARACION
   MOVER.INSECTOS
END

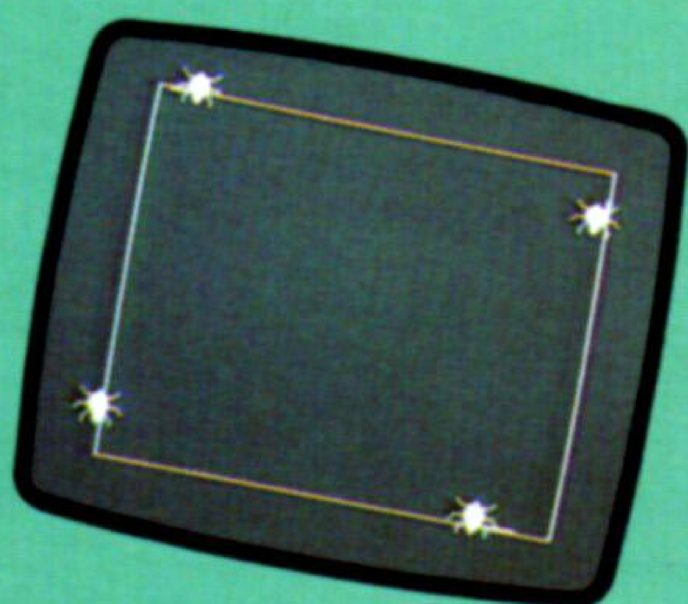
TO PREPARACION
   DRAW
   FULLSCREEN
   TELL 0
   HT
   PU
   SETXY(−100)(−100)
   CUADRADO 200
   POSICION 1(−100)(−100)
   POSICION 2(−100) 100
   POSICION 3 100 100
   POSICION 4 100(−100)
END

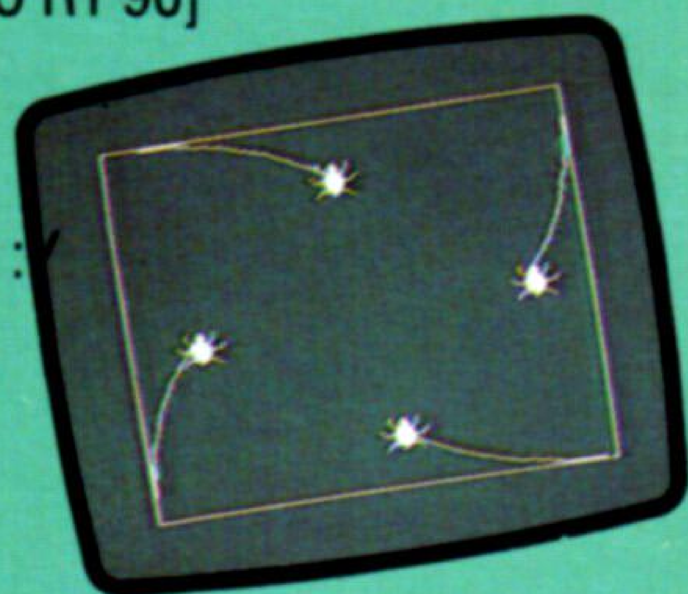
TO CUADRADO :LADO
   PD
   REPEAT 4[FD :LADO RT 90]
   PU
END

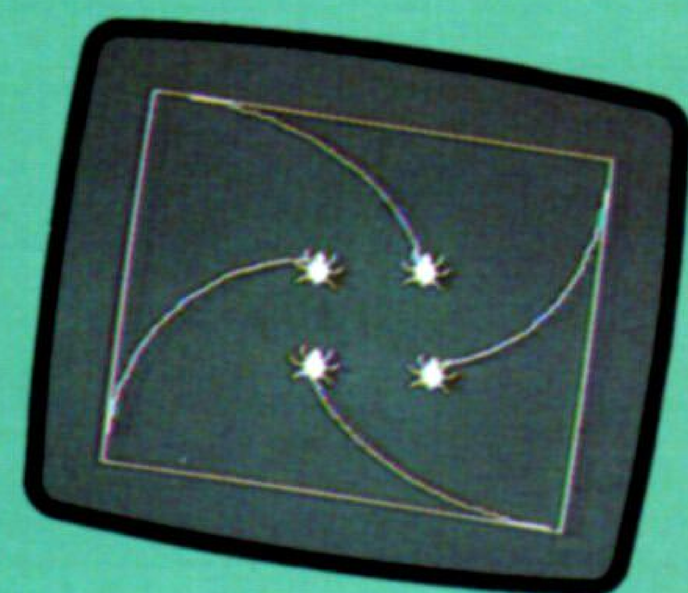
TO POSICION :NUM :X :Y
   TELL :NUM
   SETSHAPE 3
   PU
   SETXY :X :Y
   PD
   ST
END

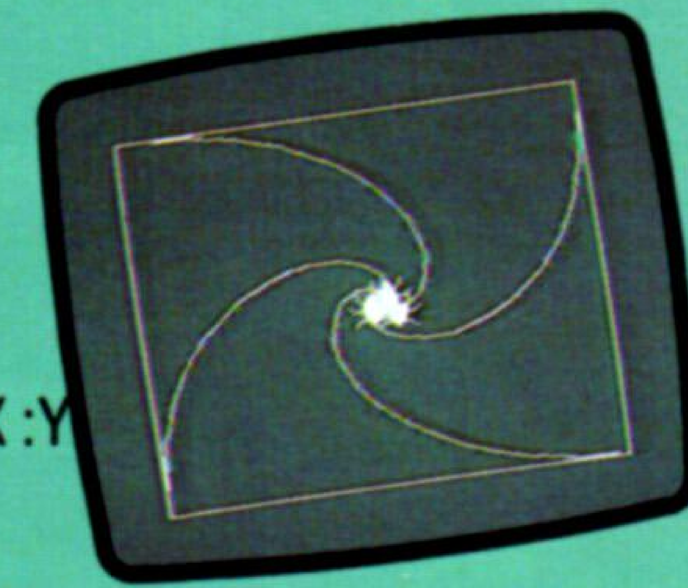
TO MOVER.INSECTOS
   SEGUIR 1 2
   SEGUIR 2 3
   SEGUIR 3 4
   SEGUIR 4 1
   MOVER.INSECTOS
END

TO SEGUIR :A :B
   TELL :B
   MAKE"X XCOR
   MAKE"Y YCOR
   TELL :A
   SETH TOWARDS :X :Y
   FD 10
END
```









Ian McKinnell

Con tres insectos

Escriba un programa para otro problema, con tres insectos situados en las esquinas de un triángulo

Proyecto Lunar Lander

Hemos desarrollado este programa como solución al propuesto en el capítulo anterior (p. 1135). Digite ALUNIZAR para participar en el juego:

```
TO ALUNIZAR
   PREPARACION JUEGO
END

TO PREPARACION
   DRAW DIBUJAR.PLATAFORMA
   PREPARAR.COHETE
END

TO DIBUJAR.PLATAFORMA
   PU SETXY(−20)(−60)PD SETXY 20(−60)PU
END

TO PREPARAR.COHETE
   SETXY 0 120 MAKE "VEL 0
   MAKE "COMBUSTIBLE 50
END

TO JUEGO
   ORDEN MOVER
   IF YCOR<−53 THEN BOOM STOP
   GRAVEDAD INFORME.COMBUSTIBLE JUEGO
END

TO ORDEN
   IF TECLALEIDA="F THEN QUEMAR
END

TO TECLALEIDA
   IF RC? THEN OUTPUT RC
   OUTPUT "
END

TO QUEMAR
   IF :COMBUSTIBLE>0 THEN MAKE "VEL
   :VEL+0.5 MAKE
   "COMBUSTIBLE :COMBUSTIBLE−1
END

TO MOVER
   SETY YCOR+:VEL
END

TO BOOM
   IF :VEL>(−1) THEN PRINT[HA HECHO UN
   BUEN ALUNIZAJE,FELICITACIONES] STOP
   PRINT [IMPACTO MATO A LA TRIPULACION!]
END

TO GRAVEDAD
   MAKE "VEL :VEL−0.2
END

TO INFORME.COMBUSTIBLE
   (PRINT"COMBUSTIBLE :COMBUSTIBLE)
END
```



Promesa oriental

Varias empresas japonesas se han puesto de acuerdo en el proyecto MSX para producir un ordenador estándar

En esta ocasión analizaremos dos de los primeros microordenadores MSX fruto del proyecto MSX: el Sony Hit-Bit y el Toshiba HX-10.

El estándar MSX (véase p. 621) determina la CPU que se utiliza (Z80), la cantidad mínima de ROM (32 Kbytes) y de RAM (8 Kbytes), el tipo de chips para gráficos y sonido, el contenido del teclado (si bien el trazado puede variar), la cantidad mínima de interfaces y su diseño; las pantallas para gráficos y para texto y, por supuesto, el lenguaje BASIC que contiene la ROM. Dado que el MSX es un diseño estandarizado, cabe esperar que todas las máquinas MSX serán similares. Los fabricantes tienen flexibilidad en cuanto a la cantidad de memoria por encima del mínimo, el tipo de teclado utilizado y la cantidad de interfaces extras. En la práctica, Sony y Toshiba, al igual que la mayoría de los fabricantes MSX, han optado por una especificación más amplia que la que dictan las exigencias mínimas.

Tanto el Sony Hit-Bit como el Toshiba HX-10 poseen teclados de buena calidad, si bien a algunos usuarios las teclas les resultarán demasiado sensibles. Los dos micros vienen con 64 Kbytes de memoria principal, y 16 Kbytes de RAM adicionales dedicados a la visualización en video. Esto da un total de 80 Kbytes, más de lo que ofrecen la mayoría de los ordenadores personales. Los modelos de Sony y Toshiba poseen cada uno una interface para impresora Centronics estándar y un par de conectores para palanca de mando, que en los ordenadores personales suelen ser opciones extras.

Originalmente se pensaba que los ordenadores MSX serían máquinas de precio reducido, pero las fluctuaciones monetarias y el aumento de los costos de producción han hecho subir los precios. Otra causa del aumento de precios ha sido la prisa por poner a la venta los ordenadores en Europa. Toshiba está vendiendo todas sus máquinas por envío aéreo, más costoso que el marítimo. La empresa ha tenido que modificar todas sus cadenas de producción, pasando de fabricar versiones japonesas de la máquina a construir el modelo europeo, con el deseo de convertirse en el primer fabricante MSX que tenga a la venta un producto en Europa.

Una de las primeras cosas que se observan al conectar un micro MSX es una fila de términos en la parte inferior de la pantalla. Éstas son palabras clave del lenguaje BASIC, como RUN, CLOAD", LIST, etc. Los micros tienen cinco teclas de función que generan estos términos tan comúnmente utilizados. Las palabras de la pantalla sirven como etiquetas para las teclas de función de modo que el usuario no tenga que recordar la función de cada tecla.

Estas teclas se definen automáticamente cuando se conecta la máquina, pero es fácil cambiar sus definiciones mediante el empleo de la instrucción KEY. A pesar de que sólo hay cinco teclas de función, se puede acceder a hasta 10 funciones pulsando la tecla Shift y la tecla de función deseada al mismo tiempo. Al pulsar Shift, las etiquetas de la pantalla cambian, pasando a reflejar las nuevas funciones asignadas a las teclas. Cada etiqueta de función puede contener hasta 15 caracteres, si bien sólo aparecerán en la pantalla los siete primeros.



Chris Stevens



Sistema estándar
El Toshiba HX-10 posee dos puertas para palanca de mando, una interface para impresora Centronics en paralelo, puerta para cartucho de ROM y "racimo" de teclas de manejo de cursor, como vemos en la fotografía. El BASIC MSX trata al control de la palanca de mando de la misma forma que el control mediante cursor, de modo que se pueden escribir juegos para un tipo de control y también utilizar automáticamente el otro tipo



El teclado y el editor de pantalla trabajan juntos para simplificar la edición. Cuatro teclas desplazan el cursor a través de la pantalla, y se pueden introducir cambios en cualquier lugar de ésta simplemente escribiendo sobre los caracteres existentes. Para insertar y eliminar caracteres se requiere la pulsación de una sola tecla. Las teclas para el cursor del Toshiba HX-10 son del mismo tamaño que las del resto del teclado, pero el Sony Hit-Bit utiliza teclas diferenciadas y grandes para su racimo de cursor. Éstas se emplean con mucha frecuencia, de modo que un diseño de esta clase puede resultar muy cómodo.

Al igual que el hardware, el software MSX está lleno de características extras. El BASIC MSX incluye instrucciones tales como AUTO y RENUMeración, y contiene varias instrucciones para generación de sonido, gráficos y tratamiento de interrupciones. Hay otras tres instrucciones fundamentales para crear gráficos. LINE dibuja una línea entre dos puntos, si bien se puede utilizar también para dibujar un cuadrado agregando la letra B (por *box*: caja) después de las coordenadas. Añadiendo las letras BF (por *box fill*: rellenar caja) se dibuja un cuadrado de color sólido. La instrucción CIRCLE se puede emplear para dibujar elipses y arcos además de círculos básicos. Y la instrucción PAINT rellenará con color sólido cualquier forma esbozada y funciona incluso con las formas más extrañas.

El BASIC MSX incluye muchas otras características útiles, aunque tal vez el juego de instrucciones más impresionante (para tratamiento de interrupciones) no se aprecie al principio. El tratamiento de interrupciones es muy útil en la programación de gráficos de alta velocidad. Existe una gran cantidad de situaciones en las que un programa debe realizar una tarea, verificando constantemente al mismo tiempo si sucede alguna otra cosa. Un ejemplo típico de esto se puede encontrar en juegos del tipo "marcianitos". El programa debe mantener a los



Enarbolando la bandera

El estándar MSX

CPU	Z80A, 3,58 MHz
RAM	Mínimo: 8 K
ROM	32 K incluyendo BASIC
PANTALLA	16 colores, gráficos de 256×192 pixels, 32 sprites, visualización de texto de 40×24 (o 32×24) (chip de video TI 9918 o equivalente)
SONIDO	3 canales, accesibles desde BASIC (chip controlador de sonido AY38910)
INTERFACES	Puerta para cartucho MSX, salida modulada para TV, impresora en paralelo Centronics, interface para cassette
TECLADO	Teclado QWERTY más teclas de función especial, 4 teclas de cursor, 10 teclas de función programables

Complementos al MSX

SONY HIT-BIT	Software de base de datos incorporado, salida RGB, paquetes opcionales RAM 4 K
TOSHIBA HX-10	Bus de ampliación, 2 puertas para palanca de mando
YAMAHA CX-5	Teclado minimúsica y software con MIDI
PIONEER	Interface controladora de videodisco
SANYO MPC100	Lápiz óptico y software opcional
JVC HC7GB	Salida RGB
SPECTRAVIDEO SVI 728	Teclado numérico completo

Si bien el estándar MSX determina un mínimo de 8 K de memoria, todos los fabricantes mencionados proporcionan 64 K de RAM para el usuario, más 16 K de RAM de pantalla

extraterrestres moviéndose por la pantalla, verificando en todo momento si se ha pulsado o no el botón de "disparo". El programa debe hacer dos cosas al mismo tiempo, pasando rápidamente de una tarea a otra.

La solución MSX consiste en designar ciertas cosas como *eventos*. Se dispone de instrucciones para decirle al ordenador que vea si ha ocurrido un evento. Cuando se produce uno, el ordenador pasa automáticamente a una subrutina para tratar el evento.

La pantalla de gráficos MSX puede visualizar 16 colores con una resolución de 256×192 pixels. Se pueden definir hasta 32 sprites de 8×8 pixels (o 16 sprites de 16×16, u 8 sprites de 32×32). Para aprovechar los sprites al máximo, el BASIC MSX incluye un juego completo de instrucciones específicas para ello, como SPRITE, para definir un sprite, y PUT SPRITE, para situar un sprite en cualquier lugar de la pantalla.

Tal como han afirmado los fabricantes MSX, ya hay muchísimo software en cartucho a la venta para las máquinas. Y la promesa de la compatibilidad parece ser verdadera: el software para el Toshiba HX-10 funciona perfectamente con el Sony Hit-Bit, y viceversa. Esto se aplica tanto al software en cartucho como a los programas en cassette. Después de años y años de sistema incompatibles, parece casi mágico sacar un cartucho de una máquina y utilizarlo en otra. Las empresas MSX se están apoyando en esta característica para sacar muy rápidamente a la venta una amplia gama de software para todas las máquinas.

Queda por ver si las máquinas MSX tendrán en el mercado el impacto que esperan los japoneses. Con la fuerte competencia encabezada por Sinclair, Commodore y Amstrad, entre otras, se avecina una enconada contienda por las ventas. Sea como fuere, las máquinas MSX responden cabalmente a las afirmaciones de sus fabricantes. Son ordenadores agradables de utilizar, están bien equipados y tienen un precio razonable.



Standard Interface



Chris Stevens

TOSHIBA HX-10 MSX

DIMENSIONES

365×245×60 mm

CPU

Z80A, 3,58 MHz

MEMORIA

64 K de RAM (28 K disponibles para BASIC), 16 K de RAM de pantalla, 32 K de ROM incluyendo BASIC

PANTALLA

40 columnas por 24 filas para texto, 256×192 para gráficos, con 16 colores y hasta 32 sprites

INTERFACES

Impresora Centronics, TV, pantalla, salida de audio, 2 puertas para palanca de mando, puerta para cassette, ranura para cartucho de ROM, bus de ampliación

LENGUAJES DISPONIBLES

Microsoft Extended BASIC

TECLADO

68 teclas tipo máquina de escribir, con racimo de cursor más 5 teclas de función programables

DOCUMENTACION

Guía de instalación y guía de referencia para programación en BASIC. Ambas están bien hechas, pero no son lo suficientemente descriptivas

VENTAJAS

El BASIC MSX posee muchas características útiles, incluyendo buenas instrucciones para gráficos y sonido. La estandarización del MSX es un dato valioso porque supondrá más software y periféricos

DESVENTAJAS

El BASIC MSX carece de la capacidad de programación estructurada; la disponibilidad de MSX y los periféricos está tardando mucho en aparecer

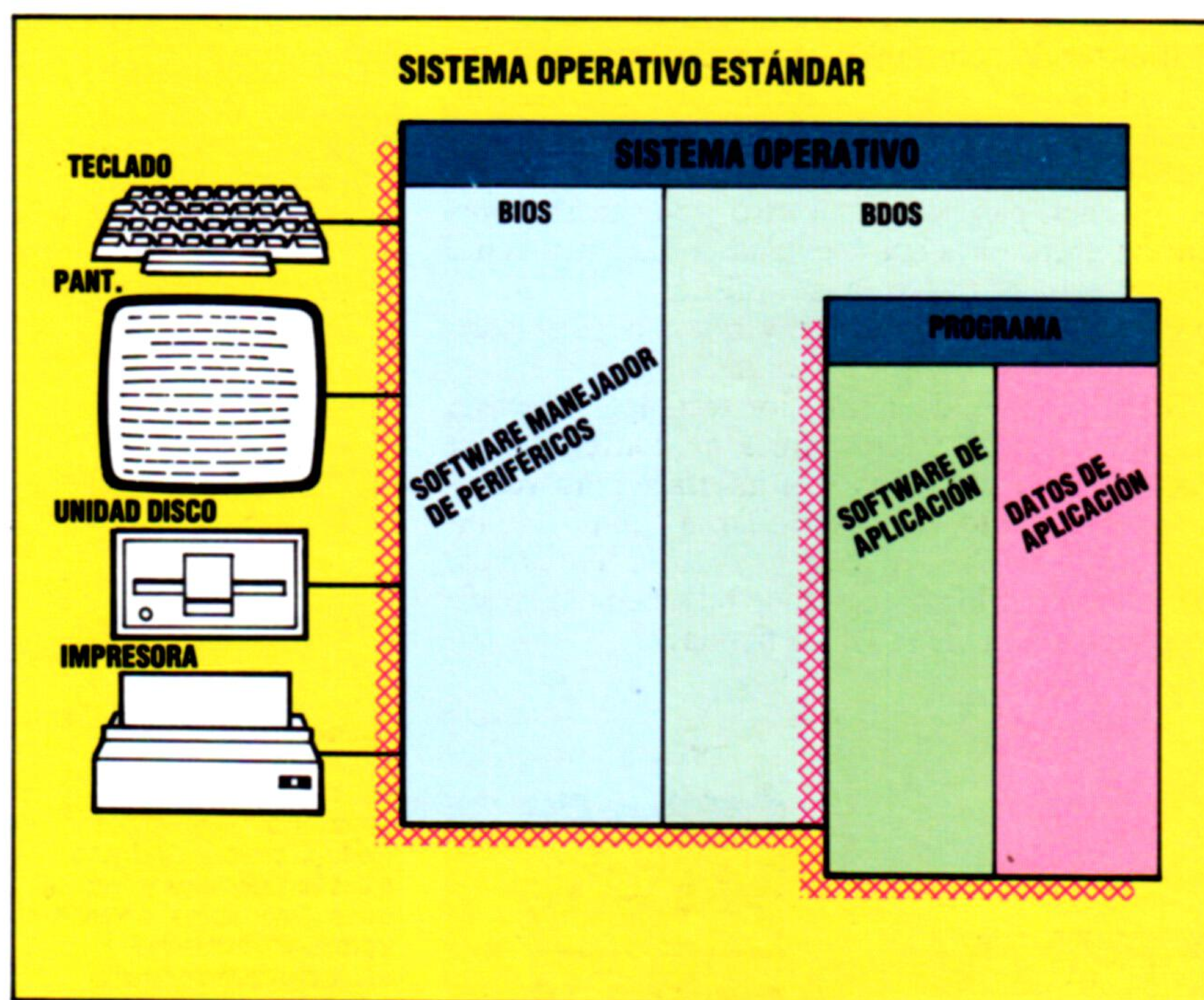
DIFERENCIAS

El Sony Hit-Bit tiene 3 programas incorporados en ROM, salida para pantalla RGB, y la disposición del teclado es ligeramente distinta

Control total

En este último capítulo sobre software integrado analizaremos algunas alternativas al programa único multifunción y de gran consumo de memoria

El enfoque alternativo al software integrado responde a un principio completamente diferente. Se basa en que el sistema operativo del ordenador proporcione las facilidades básicas de integración y que los programas individuales escritos para trabajar con ese sistema automáticamente se adapten y trabajen juntos.



Cumpliendo órdenes
Con el sistema operativo tradicional, el programa que está en ejecución tiene el mando absoluto. Su lógica determina lo que aparece en la pantalla, cuándo se ha de acceder a la unidad de disco y cuándo leer del teclado. Sus instrucciones generales pasan al sistema operativo, que administra en detalle el empleo del hardware que se está usando. El programa en ejecución es el soberano, y se da por sentada la subordinación del sistema operativo

La creación de un sistema operativo de estas características no ha sido tarea fácil, puesto que exige que el hardware y el software del ordenador sean más sofisticados que el de los diseños tradicionales. Apple ha abierto el camino con sus ordenadores Lisa y Macintosh, diseñados para el cliente, si bien hay otras empresas, en particular Microsoft, que están preparando sistemas para otros ordenadores populares, como el IBM PC.

Los programas para estos nuevos sistemas operativos son muy distintos de los programas para sistemas tradicionales. Gran número de los programas se dedican a la interface para el usuario: las rutinas que reciben órdenes e información del usuario y presentan los resultados. Las opiniones difieren en cuanto a la operatoria de los programas, de modo que casi todos los paquetes poseen sus propios procedimientos y han de ser aprendidos desde cero.

Un sistema operativo integrado proporciona un juego incorporado de rutinas de interface para el usuario para que lo utilicen todos los programas de aplicaciones. Cuando un programa desea visualizar una lista de opciones en la pantalla para que el usuario realice su elección entre ellas, debe usar para ello una rutina ya incorporada en el sistema operativo. La ventaja que esto ofrece reside en que todos los programas escritos para trabajar con el mismo sistema operativo tendrán prácticamente los mismos procedimientos de operatoria. Una vez que el usuario ha aprendido a utilizar un programa en el sistema, ¡está listo para emplear el resto de programas disponibles!

Una interface para el usuario proporcionada en particular para estos programas es el ratón. Éste es un dispositivo señalador que se emplea para seleccionar opciones de la pantalla a través de un cursor no tradicional. Una alternativa es la *pantalla sensible al tacto*, en la que una matriz de haces luminosos responde ante el toque de un dedo. La visualización se divide en "ventanas" separadas, cada una de las cuales contiene una opción o tarea diferente. Técnicamente, una interface para el usuario de este tipo requiere un procesador rápido, muchísima memoria y gráficos de gran resolución. Pero estos costos adicionales bien valen la pena, porque el sistema, por lo general, es aplicable a casi cualquiera de los programas disponibles, es muy fácil de aprender y proporciona la forma más sencilla posible de que el usuario sea capaz de ver varias aplicaciones a la vez y pasar de una a otra.

Control de operaciones

Es importante apreciar la forma en que este sistema integra los programas. El programa y el usuario no están nunca en contacto directo: todo se ha de hacer a través del sistema operativo, y es éste el que tiene el control en todo momento. De hecho, cada programa de aplicaciones se convierte en una extensión del sistema operativo y el ordenador es un único "entorno" integrado.

Esto nos lleva a la segunda diferencia fundamental en la forma en que funcionan esta clase de sistemas. En un sistema tradicional, la comunicación entre programa y sistema operativo es en gran parte unidireccional. El programa solicita que se lleve a cabo una tarea específica y el sistema operativo la realiza en consecuencia.

En un sistema integrado, el sistema operativo está al mando y a él le corresponde solicitar cosas al programa. Por ejemplo, el sistema operativo puede enviarle al programa un mensaje que diga "¿Podrías volver a dibujar tu visualización, ya que el usuario la ha desplazado a otra zona de la pantalla?"; o "Manténlo todo; el usuario ha llevado el ratón a una aplicación distinta"; o "Aquí hay algunos datos para ti tomados de una hoja electrónica". O sea, el programa ha de ser capaz de responder a las exigencias y las demandas del sistema operativo, contrariamente a los sistemas tradicionales.



Una vez que se tiene este grado de cooperación entre todo el software de una máquina, es fácil construir un entorno integrado. Cada programa posee su propia ventana en la pantalla. Cuando el usuario coloca el ratón dentro de la ventana y selecciona una opción, el sistema operativo lo notifica a este programa determinado y se lleva a cabo la operación correspondiente.

Por ejemplo, si el usuario se mueve hasta la esquina de la ventana y selecciona la opción para seleccionar esa ventana y desplazarla a una nueva posición, las rutinas del sistema operativo llevan a cabo la tarea y luego, de ser necesario, informan al programa sobre los cambios, de modo que pueda modificar su visualización adecuadamente. Si el usuario lleva al ratón hasta una ventana diferente, el programa original queda temporalmente suspendido y el sistema operativo comienza a trabajar con el nuevo programa: pasar de una aplicación a otra es tan sencillo como desplazar el ratón.

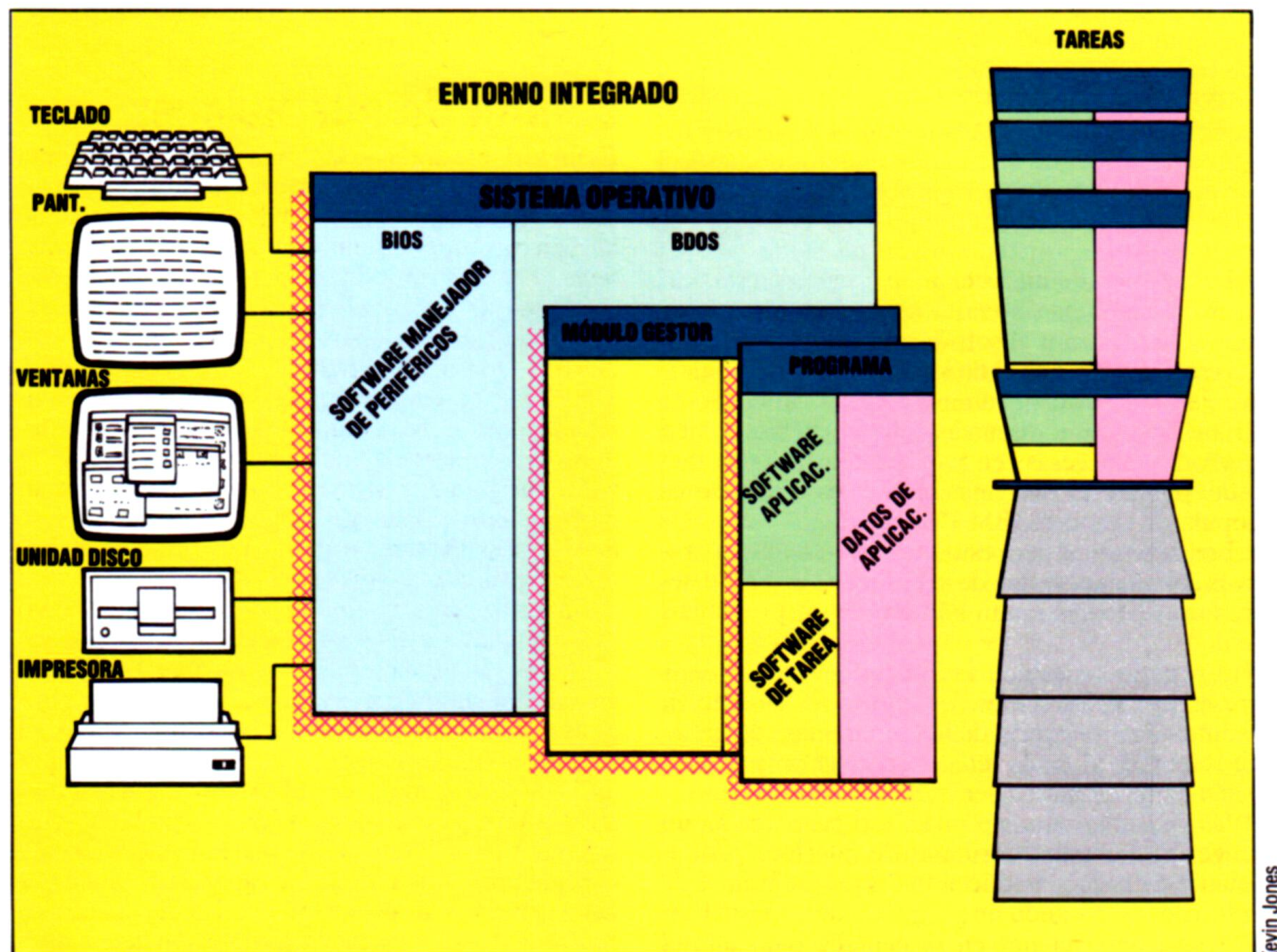
Al igual que los grandes programas "todo en uno", tales sistemas requieren que los programas y la información que sale en la pantalla en un momento dado estén en la memoria y listos para utilizar. Para facilitar esto, muchos sistemas poseen memorias masivas: un Megabyte en el Apple Lisa, por ejemplo, y 512 Kbytes en el Macintosh. Aun así, por lo general es necesario que el sistema operativo ocasionalmente deba intercambiar información y programas entre memoria y discos para acomodar todo. Para conseguir una rapidez aceptable, suele ser necesario operar en un disco rígido.

Con el objeto de que los datos se puedan intercambiar fácilmente entre los programas, el sistema operativo posee un conjunto de formatos y rutinas incorporado para transferir información. Cuando se "exportan" algunos datos de un programa y se requiere "importarlos" a otro, el sistema operativo suspende el primer programa e inicia el segundo; luego le solicita a la aplicación en curso que lea y procese la información proveniente de otro programa. Estos caminos se pueden preparar automáticamente, de modo que cuando se modifica la información de una hoja electrónica, por ejemplo, también se modificará de manera automática una gráfica de la misma hoja electrónica. Los dos programas no se ejecutarán al mismo tiempo: el sistema operativo simplemente hace malabarismos entre los dos en la medida de lo necesario.

El Apple, de Lisa, nos presenta una opción ligeramente más sofisticada, en que la información puede ser reducida a una ventana tipo "tablilla de anuncios" desde cualquier programa y luego ser "pegada" en otro. La información de formateo es transportada junto con los datos, de modo que un gráfico creado con software de gráficos de gestión es transferido como tal a otro programa.

Ésta es, por tanto, la forma más acertada de crear software integrado. Permite que el usuario mezcle y empareje cualesquiera de los programas del sistema, pase de uno a otro y transfiera información entre ellos con facilidad. El inconveniente es que requiere un hardware sofisticado que, por el momento, es bastante caro y que hay muy poco software disponible para integrar.

No obstante, toda innovación tecnológica de esta magnitud llevará tiempo antes de convertirse en algo cotidiano. El ratón y la interface para ventanas, por ejemplo, los desarrollaron equipos de investigación de Xerox hace ya más de diez años, ¡pero ha sido sólo ahora cuando tal sistema ha podido ponerse a la venta en las tiendas!



Operaciones combinadas
En un sistema integrado, el sistema operativo se ve mejorado por la adición de un módulo "gestor", que trata a todos los programas y datos en curso como "tareas" a planificar y procesar y manipula el detallado sistema operativo subyacente como un simple software de apoyo al sistema. Este módulo coloca y elimina las tareas en la memoria principal y en los discos de acuerdo a los requerimientos del usuario y las necesidades de las tareas en curso. Está equipado para pasar la información entre las aplicaciones en formatos estándares y, por tanto, posibilita la transferencia de datos entre las tareas. De hecho, el gestor es en sí mismo una tarea de alta prioridad, y su relación con las otras tareas es simbiótica en vez de servil

Dando la alarma

He aquí algunos programas para utilizar la caja de relés en sencillas aplicaciones domésticas

La caja de relés está diseñada para controlar el suministro de electricidad a cualquier dispositivo que esté conectado a ella. En respuesta a una señal de bajo voltaje, la caja abre o cierra la alimentación de corriente de la red al conector montado en la caja. El modo de operación es tal que la alimentación de la red al conector se mantiene mientras haya una corriente de poco voltaje suministrada al relé. Por consiguiente, podemos activar éste directamente desde la caja de salida de bajo voltaje que construimos previamente (véase p. 1054). El suministro de energía eléctrica de la red desde la caja de relés reflejará exactamente la corriente de bajo voltaje proporcionada al relé desde la caja de salida de bajo voltaje. Por lo tanto, se puede conseguir el control de la alimentación de red mediante las mismas técnicas de software utilizadas para controlar los dispositivos de bajo voltaje.

Si, por ejemplo, se conectan los cables de bajo voltaje del relé a las conexiones positiva y negativa de la línea 0 de la caja de salida, y se enchufa en un conector de corriente, se le suministrará corriente al conector de la caja de relés cuando el bit 0 del registro de datos de la puerta para el usuario se envíe alto. Siempre que el bit 0 se envíe bajo se interrumpirá el suministro de electricidad al conector de la caja de relés. Hasta cuatro cajas de relés o disyuntoras se pueden conectar a la caja de salida de bajo voltaje y conmutar de este modo.

Podemos valernos de esta sencilla disposición de conmutación para desarrollar unos cuantos sistemas de control que utilicen los aparatos domésticos de uso cotidiano. Primero probaremos un proyecto sencillo, en el cual utilizamos una grabadora de cinta para programar un micro de modo que responda "verbalmente" a la presión sobre un teclado.

En primer lugar, necesitamos grabar una serie de frases, tales como "Estás pisoteando mi teclado", seguido de "Lo has vuelto a hacer" y "Oye, ¡te lo he advertido!", y así sucesivamente. Después de grabados los mensajes, conectaremos el teclado y la grabadora al sistema de la puerta para el usuario y escribiremos un poco de software para activar las frases, de una en una, en respuesta a una presión repetida sobre el teclado.

En el sistema de la puerta para el usuario hemos de hacer las siguientes conexiones:

1) Enchufar los cables de voltaje del relé o disyuntor de red en los terminales positivo y negativo de la línea 0 de la caja de salida de bajo voltaje.
2) Enchufar el cable de alimentación a la caja de relés en un enchufe de pared.
3) Conectar los dos cables del teclado de presión a través de los terminales positivo y negativo de la línea 7 de la caja buffer.

El principal problema que supone el diseño de software para este sistema es asegurar que la grabadora se encienda y se apague con exactitud cuando se reproduce un mensaje. Por consiguiente, antes de que podamos escribir un programa debemos cronometrar con suma precisión cada mensaje e introducir estos datos en el programa controlador. El cronometraje se puede realizar utilizando el reloj interno del micro o un cronómetro. Si en la cinta hay tres frases que duran períodos de T(1), T(2) y T(3) segundos, entonces podemos escribir un programa que, al activarse desde el teclado de presión, encienda la grabadora de cinta durante el período de tiempo correcto para cada uno de los sucesivos mensajes. Si el cronometraje de las frases se efectúa con precisión, entonces cada frase estará justo en su inicio cuando se encienda la grabadora.

Los siguientes programas (para el Commodore 64 y el BBC Micro) activarán la grabadora para tres intervalos de tiempo sucesivos T(1), T(2) y T(3) en respuesta a impulsos provenientes del teclado de presión. Estas variables deben ser inicializadas con los valores correspondientes a las tres frases elegidas.

BBC Micro

```
10 REM PROGRAMA PISOTEANDO BBC
20 DIM T(3)
30 RDD=&FE62:REGDAT=&FE60
40 ?RDD=127:REM L7 ENTRADA
50 ?REGDAT=0:REM TODAS APAGADAS
60 CLS
70 FORI=1TO3
80 INPUT"INTERVALO DE TIEMPO (SEGS)";T(I)
90 NEXT I
100 :
110 FORL=1TO3
120 CLS
130 REPEAT
140 UNTIL(?REGDAT AND 128)=0:REM L7 BAJA
150 ?REGDAT=1:REM CONECTAR CINTA
160 TIME=0:REM INICIAR RELOJ
170 REPEAT
180 UNTIL TIME>T(L)*100
190 ?REGDAT=0:REM APAGAR CINTA
200 NEXT L
210 END
```

Commodore 64

```
10 REM PROGRAMA PISOTEANDO CBM 64
20 DD=56579:REGDAT=56577
30 POKERDD,127:REM L7 ENTRADA
40 POKEREGDAT,0:REM TODAS APAGADAS
50 PRINTCHR$(147): REM LIMPIAR PANTALLA
60 FOR I=1 TO 3
70 INPUT"INTERVALO DE TIEMPO (SEGS)";T(I)
80 NEXT I
90 :
100 FOR L=1TO3
110 IF(PEEK(REGDAT)AND128)<>0 THEN 110
115 POKE REGDAT,1:REM ENCENDER CINTA
120 T=TI:REM INICIALIZAR RELOJ
130 IF T(L)>(TI-T)/60 THEN130
140 POKEREGDAT,0:REM APAGAR CINTA
150 NEXT L
160 END
```



Reloj despertador

Habiendo desarrollado un sistema sensible a las pisadas sobre un teclado de intrusos, vamos ahora a considerar un proyecto para convertir un micro en un cómodo reloj despertador programable. Un sistema de este tipo se puede, por supuesto, confeccionar a la medida exacta de las necesidades de cada uno. El programa que ofrecemos (en versiones para el Commodore 64 y el BBC Micro) permite que el usuario entre:

1) la hora del día;
2) el número de intervalos "para cabecear" (períodos entre los estallidos del zumbador o música) requeridos;
3) si para cada intervalo para cabecear se requiere un período de música, de alarma o de silencio, y la longitud del intervalo;
4) si para cada intervalo se debe encender o no una luz;
5) la última hora posible para levantarse.

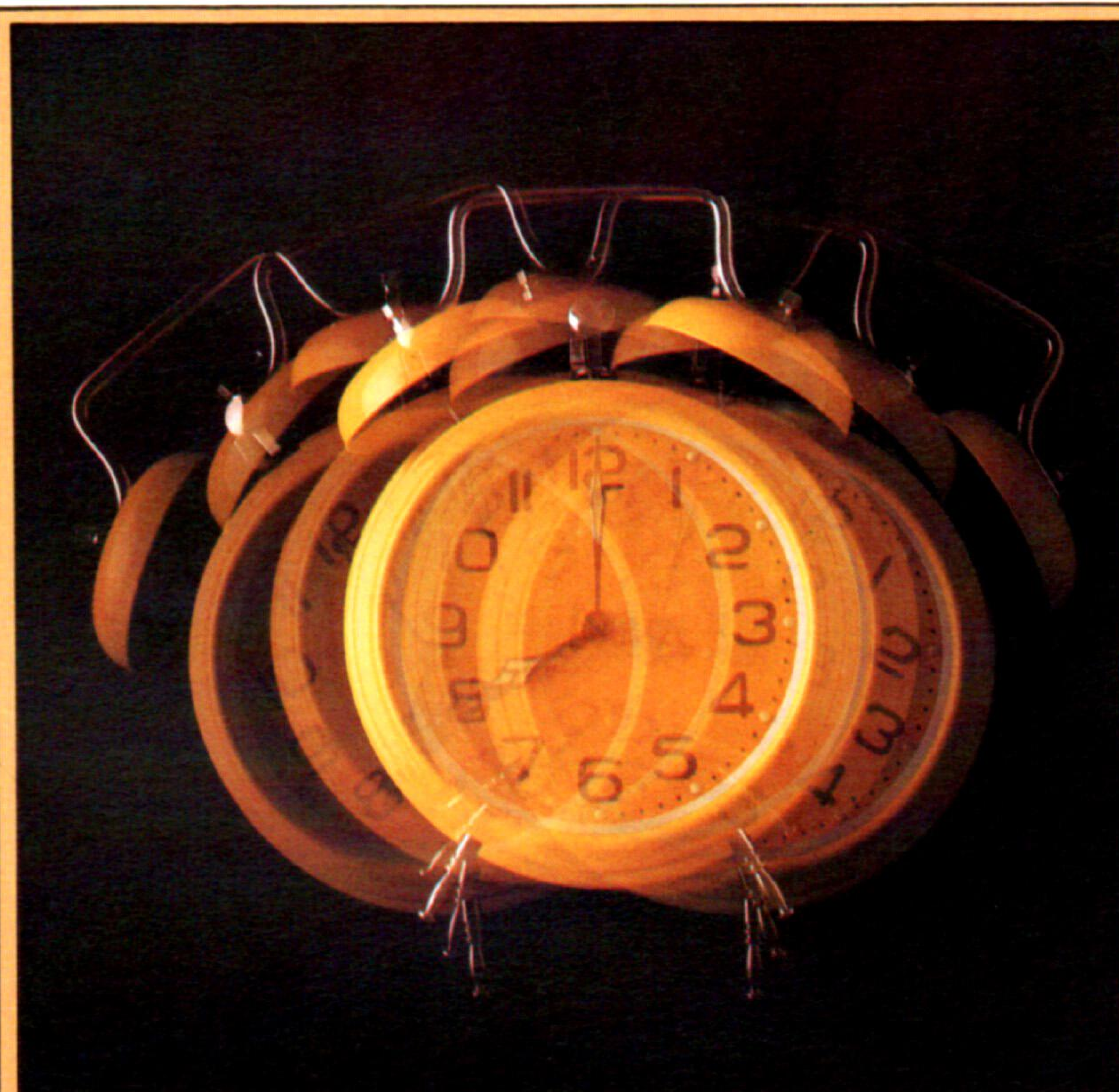
El programa se basa en la presunción de que se realizan las siguientes conexiones a la caja de salida de bajo voltaje:

1) Se conecta una grabadora de cinta a la línea 0 a través de un relé o disyuntor de red.
2) Se conecta una lamparilla de mesa a la línea 1 a través de un relé en la línea 1.
3) Se conecta directamente a la línea 3 un timbre eléctrico de 9 voltios.

El programa acepta la última hora para levantarse y cuenta hacia atrás el número de intervalos programados para calcular la hora de comienzo de cada intervalo. Se utilizan matrices para almacenar los datos que nos dicen qué aparatos han de estar encendidos durante cada período. Observe que a las variables de matriz se les asignan valores que corresponden al valor de bits requerido en el registro de datos para encender cada aparato en particular. Mediante el empleo de la instrucción lógica OR podemos simplemente hallar el total compuesto que se debe colocar en el registro de datos para activar cualquier combinación de dispositivos.

La mayor parte de nuestro esfuerzo de programación se ha dirigido hacia la manipulación de variables en serie (*strings*) para permitir realizar cálculos numéricos. Esto es especialmente cierto para el programa del Commodore 64, porque la versión de BASIC que utiliza esta máquina carece de las útiles instrucciones MOD y DIV de que disponen los programadores del BBC Micro.

Ahora hemos desarrollado un sistema de entrada y salida verdaderamente flexible para control por microordenador, que nos permite controlar LED, dispositivos de bajo voltaje y aparatos que funcionan con la red eléctrica, además de permitir que el micro acepte e interprete datos entrados desde una gama de sensores. Ahora se nos abren muchas posibilidades en el diseño de sistemas de control para que los empleemos nosotros mismos. En los ejemplos que ofrecemos el micro se utiliza como un sofisticado reloj programable. Otras aplicaciones implicarían encender y apagar estufas eléctricas en respuesta a un par de sensores de calor, o encender una bombilla eléctrica por la noche. Las posibilidades para la experimentación son infinitas.



Ian McKinnell

Commodore 64

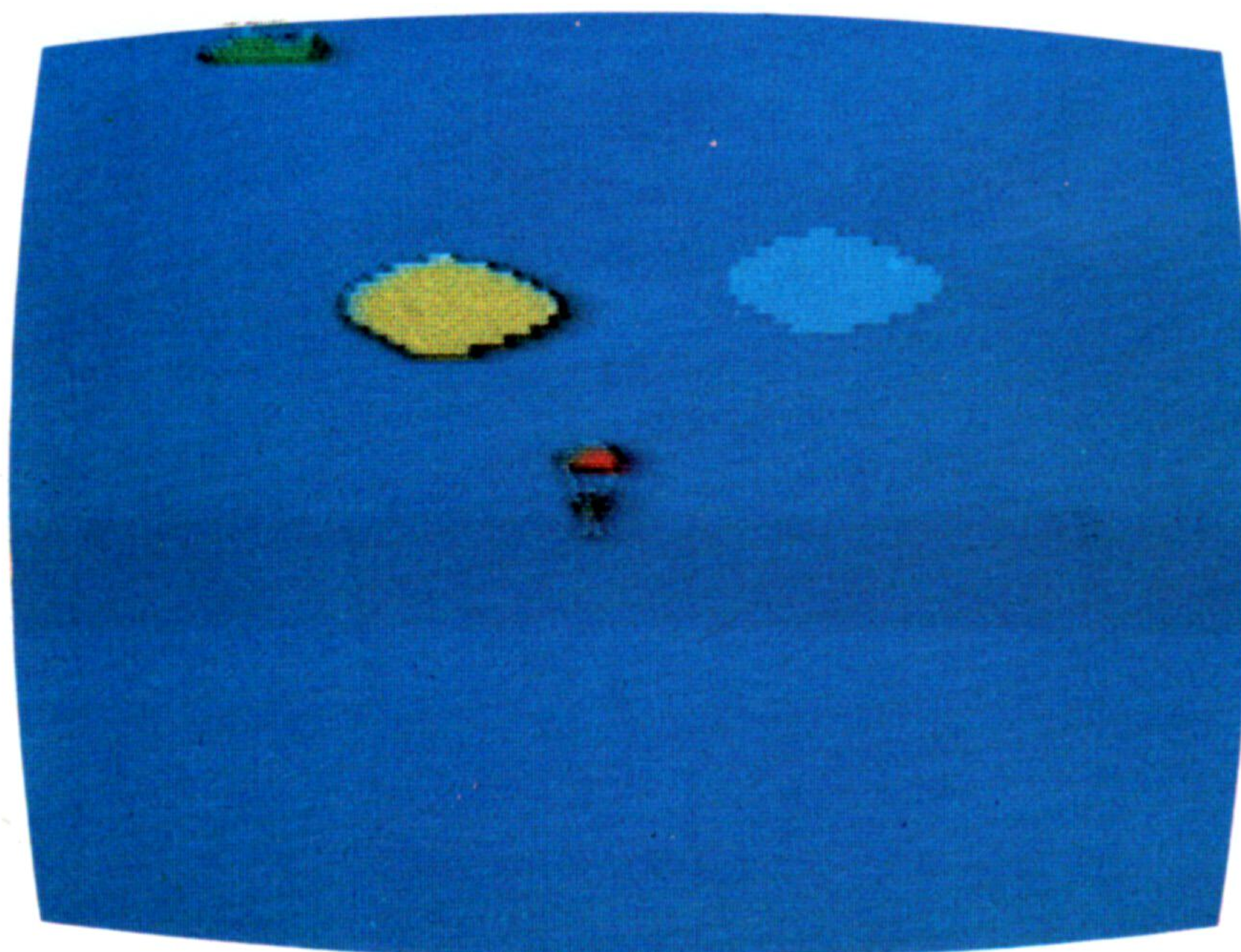
```
100 REM**** RELOJ DESPERTADOR CBM 64 ****
110 RDD=56579:REGDAT=56577
120 POKE RDD,255:POKEREGDAT,0
130 PRINTCHR$(147):REM LIMPIAR PANTALLA
140 INPUT"NUMERO DE INTERVALOS PARA CABECEAR";N
150 M=N+1
160 DIM A(M),M(M),L(M),TS$(M),T(M)
170 :
180 REM**** ENTRAR DATOS INTERVALOS ****
190 FOR C=1 TO N
200 PRINT:PRINT"NUMERO DE INTERVALO";C
210 INPUT"MUSICA,ALARMA O SILENCIO (M/A/S)";AN$
215 AN$=LEFT$(AN$,1)
220 IF AN$<>"M"ANDAN$<>"A"ANDAN$<>"S" THEN 210
230 IF AN$="M" THEN M(C)=1:A(C)=0
240 IF AN$="A" THEN A(C)=8:M(C)0
250 IF AN$="S" THEN A(C)=0:M(C)=0
260 INPUT"LUZ ENCENDIDA (S/N)";AL$
270 L$=LEFT$(AL$,1)
280 IF L$<>"S" AND L$<>"N" THEN 260
290 IF L$="S" THEN L(C)=2:GOTO310
300 L(C)=0
310 INPUT"INTERVALO DE TIEMPO (MIN)";T(C)
320 NEXT C
330 :
340 INPUT"HORA MAXIMA PARA LEVANTARSE (HHMM)";LT$
350 LT$=LT$+"00":REM AGREGAR SEGUNDOS
360 TS$(N+1)=LT$:REM HORA MAXIMA
370 REM CONVERTIR HORA MAXIMA A MINUTOS
380 LM=60*VAL(LEFT$(LT$,2))+VAL(MID$(LT$,3,2))
390 :
400 INPUT"AHORA SON LAS (HHMM).";TN$
410 TI$=TN$+"00": REM INICIAR RELOJ
420 :
430 REM**** ANALIZAR Y CALCULAR ****
440 REM ****** CALCULAR HORAS COMIENZO INTERVALOS ******
450 FOR C=N TO 1 STEP -1
460 LM=LM-T(C):REM HORA COMIENZO EN MIN
470 HR=INT(LM/60)
480 MN=INT(60*(LM/60-HR+.000001))
490 HR$=STR$(HR):REM HORAS
500 MN$=STR$(MN):REM MINUTOS
510 MN$=MID$(MN$,2,LEN(MN$))
520 HR$=MID$(HR$,2,LEN(HR$))
530 REM** AGREGAR CEROS POR DELANTE **
540 SP$="00"
550 HR$=LEFT$(SP$,2-LEN(HR$))+HR$
560 MN$=LEFT$(SP$,2-LEN(MN$))+MN$
570 TS$(C)=HR$+MN$+"00"
580 NEXT C
590 :
600 REM** LISTO **
610 PRINTCHR$(147)
620 FOR C=1 TO N+1
630 IF TI$<TS$(C)THENGOSUB710:GOTO630
640 DN=M(C) OR A(C) OR L(C):REM DATOS REGDAT
650 POKE REGDAT,DN
670 NEXT C
680 POKE REGDAT,0
690 END
700 :
710 REM**** S/R VISUALIZAR RELOJ ****
720 PRINTCHR$'145);:REM CURSOR ARRIBA
730 PRINTLEFT$(TI$,2);":";MID$(TI$,3,2);
740 PRINT"    ";RIGHT$(TI$,2)
750 RETURN
```

BBC Micro

```
10 REM RELOJ DESPERTADOR BBC
15 MODE7
20 RDD=&FE62:REGDAT=&FE60
30 CLS
40 INPUT"CANTIDAD DE INTERVALOS PARA CABECEAR";N
45 M=N+1
50 DIM A(M),M(M),L(M),T(M),TS(M)
70 REM**** INPUT DATOS INTERVALOS ****
80 FORC=1 TO N
90    PRINT"NUMERO DE INTERVALO";C
96    REPEAT
100      PRINT "MUSICA,ALARMA O SILENCIO":
103      INPUT"  (M/A/S)";AN$
105      AN$=LEFT$(AN$,1)
110      UNTIL AN$="M"ORAN$="A"ORAN$="S"
120   IF AN$="M"THEN M(C)=1:A(C)=0
130   IF AN$="A" THEN M(C)=0:A(C)=8
140   IF AN$="S" THEN M(C)=0:A(C)=0
150   REPEAT
160      INPUT"LUZ ENCENDIDA (S/N)";AL$
170      AL$=LEFT$(AL$,1)
180      UNTIL AL$="S" OR AL$="N"
190   IF AL$="S" THEN L(C)=2 ELSE L(C)=0
200   INPUT"INTERVALO TIEMPO (MIN)":T(C)
210   NEXT C
220 :
230 INPUT"HORA MAXIMA PARA LEVANTARSE (HHMM)";LT$
232 TS(N+1)=6000*(60*VAL(LEFT$(LT$,2)))
234 TS(N+1)=TS(N+1)+VAL(RIGHT$(LT$,2))
236 REM CONVERTIR HORA MAXIMA A MINUTOS
237 LM=60*VAL(LEFT$(LT$,2))
239 LM=LM+VAL(RIGHT$(LT$,2))
240 INPUT"AHORA SON LAS (HHMM)";TN$
250 TIME=6000*(60*VAL(LEFT$(TN$,2)))
255 TIME=TIME+VAL(RIGHT$(TN$,2))
260 :
270 REM ANALIZAR Y CALCULAR
280 FORC=N TO 1 STEP -1
290   LM=LM-T(C):REM COMIENZO INTERVALOS
300   TS(C)=6000*LM
310   NEXT C
320 :
330   REM**** LISTO ****
340 CLS
350 FOR C=1 TO N+1
360   REPEAT
370      PROCreloj
380      UNTIL TIME>=TS(C)
390   DATOSREG=M(C)OR A(C) OR L(C)
400   ?REGDAT=DATOSREG
420   NEXT C
430 ?REGDAT=0
440 END
999 :
1000 DEF PROCreloj
1020 MIN=(TIME DIV 6000) MOD 60
1030 HR=(TIME DIV 6000) MOD 60
1040 MIN$=STR$(MIN):HR$=STR$(HR)
1042 REM AGREGAR CEROS POR DELANTE
1043 SP$="00"
1044 HR$=LEFT$(SP$,2-LEN(HR$))+HR$
1045 MIN$=LEFT$(SP$,2-LEN(MIN$))+MIN$
1050 PRINTTAB(18,12)HR$;":";MIN$
1060 ENDPROC
```

Paracaídas

Iniciamos un nuevo apartado, en el que proporcionaremos listados de juegos. He aquí "Paracaídas", para el Commodore 64



Saltando de un helicóptero en vuelo, intente alcanzar el blanco situado en el suelo. Una primera presión sobre una tecla le permitirá bajar verticalmente en caída libre. Una segunda presión abrirá el paracaídas. El descenso continuará más lentamente, en un ángulo de 45°, por el empuje del viento. Cuanto más espere a abrir el paracaídas, menor será la desviación. Pero no aguarde demasiado, ya que por debajo de los 100 m éste no se abrirá.

```
5 REM*********************************************
10 REM*                PARACAIDAS                  *
15 REM*********************************************
20 PRINT CHR$(147)
25 GOSUB 10000
30 POKE V+5,50
40 POKE V+1,100
50 POKE V+3,95
70 AV=320
80 N2=320
85 N1=0
90 MS=6
100 POKE 2040,14
110 POKE 2041,14
115 POKE 2042,13
120 POKE 2043,15
125 POKE 2044,16
130 POKE 2045,17
135 POKE V+23,3
140 POKE V+29,3
150 POKE V+39,1
160 POKE V+40,14
170 POKE V+41,5
200 POKE V+42,10
210 POKE V+43,10
215 POKE V+44,5
220 VV=4
230 NN=0
240 SA=0
450 POKE V,N1
460 POKE V+2,63
470 POKE V+4,63
480 POKE V+16,MS
490 POKE V+21,39
500 AV=AV-2
510 IF AV<1 THEN AV=320:MS=MS+4
520 IF AV=254 THEN MS=MS-4
540 N1=N1+1
550 IF N1>320 THEN N1=0:MS=MS-1
560 IF N1=256 THEN MS=MS+1
580 N2=N2-1
590 IF N2<1 THEN N2=320:MS=MS+2
600 IF N2=255 THEN MS=MS-2
610 H=H+VV
620 IF H>255 THEN H=255
630 IF H=230 AND SA>0 THEN 5000
1000 POKE V+16,MS
1010 IF N1<256 THEN POKE V,N1:GOTO 1030
1020 POKE 2043,15
1025 POKE 2044,16
1030 IF N2<256 THEN POKE V+2,N2:GOTO 1050
1040 POKE V+2,N2-255
1050 IF AV<256 THEN POKE V+4,AV:GOTO 1070
1060 POKE V+4,AV-255
1070 GET X$
1080 IF X$="" OR AV>255 THEN 1500
1090 ON SA GOTO 1300,1500
1100 SA=1
1110 P=AV
1120 POKE V+6,P
1130 POKE V+21,47
1140 H=58
1150 GOTO 1500
1300 IF H>150 THEN 1500
1305 POKE V+21,55
1310 VV=1
1320 NN=1
1340 SA=2
1500 P=P-NN
1510 IF P<1 AND SA>0 THEN 6000
2000 POKE V+7,H
2010 POKE V+9,H
2020 POKE V+6,P
2030 POKE V+8,P
2040 GOTO 500
5000 IF ABS(L-P)>4 OR SA=1 THEN 6000
5010 SC=SC+10
5020 FOR I=1 TO 1000
5030 NEXT I
5040 RESTORE
5050 GOTO 20
6000 POKE V+21,0
6010 PRINT CHR$(147)
6020 FOR I=1 TO 10
6030 PRINT
6040 NEXT I
6050 PRINT TAB(13)"PUNTOS[1SPC]:";SC
6060 PRINT
6070 PRINT
6080 PRINT TAB(13);"OTRA[1SPC]?"
6090 GET X$
6100 IF X$="" THEN 6090
6110 IF X$<>"N" THEN RUN
6120 END
10000 GOSUB 30000
10010 FOR I=0 TO 29
10020 READ Q
10030 POKE 832+I,Q
10040 NEXT I
10050 FOR I=30 TO 62
10060 POKE 832+I,0
10070 NEXT I
10080 FOR I=0 TO 32
10090 READ Q
10100 POKE 896+I,Q
10110 NEXT I
10120 FOR I=33 TO 62
10130 POKE 896+I,0
10140 NEXT I
10150 FOR I=0 TO 62
10160 READ Q
10170 POKE 960+I,Q
10180 NEXT I
10190 FOR I=0 TO 62
10200 READ Q
10210 POKE 1024+I,Q
10220 NEXT I
10230 FOR I=0 TO 53
10240 POKE 1088+I,0
10250 NEXT I
10260 FOR I=18 TO 20
10270 POKE 1088+I*3,255
10280 POKE 1088+I*3+1,248
10290 POKE 1088+I*3+2,0
10300 NEXT I
10310 RETURN
20000 DATA 0,0,0,127,255,0,0,128,0
20005 DATA 1,192,7,3,240,7,31,252,15
20010 DATA 127,255,255,255,255,255
20020 DATA 255,255,255,127,255,254
20030 REM
20040 DATA 0,112,0,7,255,0,31,255,224
20050 DATA 127,255,248,255,255,254
20060 DATA 255,255,255,255,255,255
20070 DATA 127,255,254,31,225,248
20080 DATA 7,255,192,1,252,0
20090 REM
20100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
20110 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
20120 DATA 13,128,0,5,0,0,5,0,0,7,0,0
20130 DATA 7,0,0,7,0,0,31,192,0,18,64,0
20140 DATA 23,64,0,7,0,0,0,0,0,0,0,0
20160 REM
20170 DATA 15,128,0,63,224,0,127,240,0
20180 DATA 127,240,0,255,248,0,255,248,0
20190 DATA 64,16,0,64,16,0,32,32,0
20200 DATA 32,32,0,23,64,0,23,64,0
20210 DATA 18,64,0,31,192,0,7,0,0,7,0,0
20220 DATA 7,0,0,5,0,0,5,0,0,13,128,0
30000 V=53248
30005 POKE V+21,0
30010 POKE V+11,230
30020 L=INT(RND(I)*100)+51
30030 POKE V+10,L
30040 RETURN
```

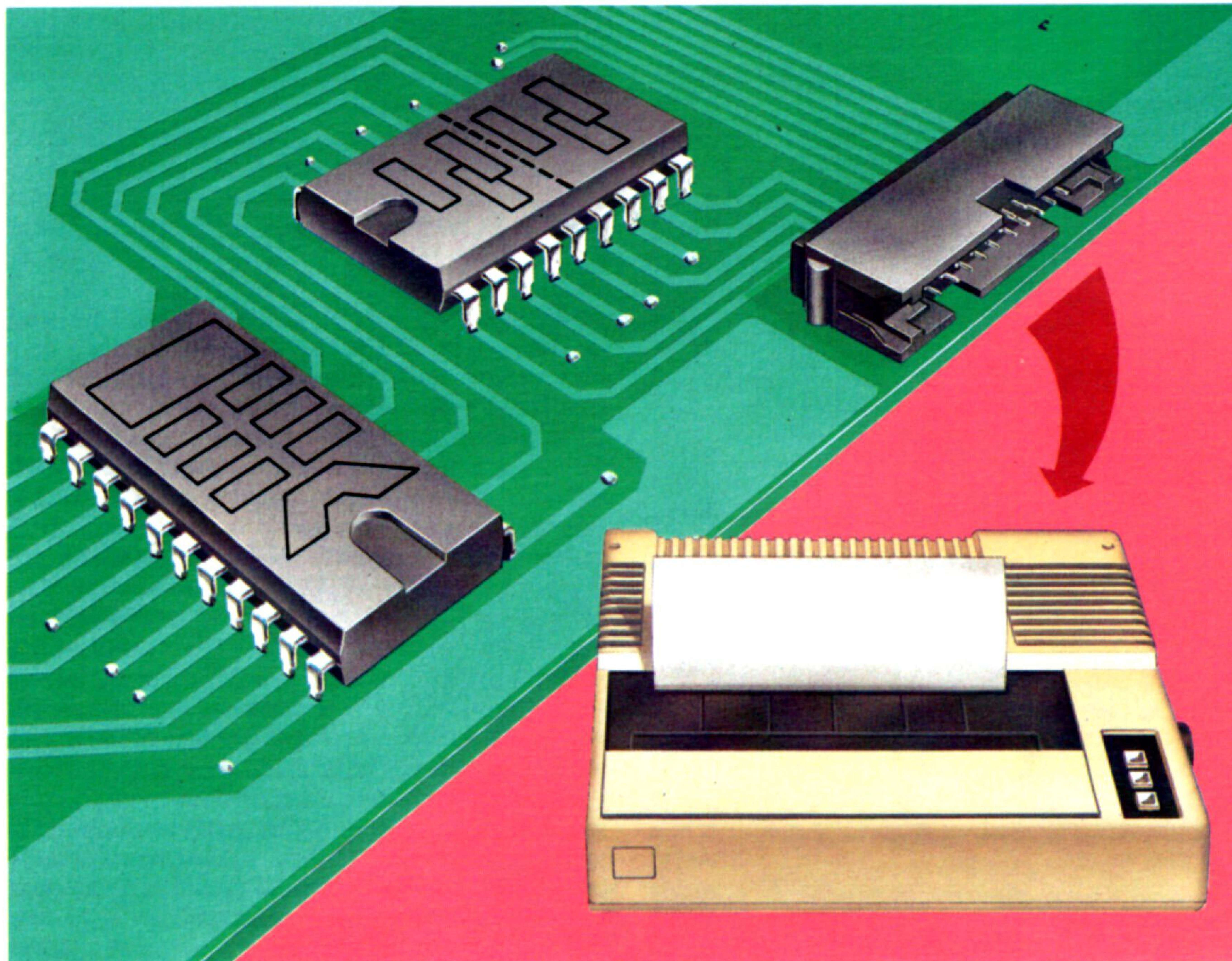
Puños fuera

Entre los aspectos más importantes de la programación en lenguaje assembly se halla el control de las entradas y salidas

El procesador 6809, al igual que el 6502 y a diferencia del Z80, no dispone de un espacio aparte de direcciones de E/S, ni tampoco de instrucciones particulares de E/S. En su lugar, los chips de la interface encargada del dispositivo de entrada/salida están asentados en la zona de direcciones normales y son manejados por medio de instrucciones de acceso a la memoria. Estos dispositivos son considerados por el procesador como otras tantas posiciones de memoria exactamente igual que las demás. La ventaja de este sistema es que resulta sencillo y rápido, y la desventaja es que se "roba" un bloque de direcciones a la memoria inutilizable en consecuencia para fines normales. Lo que significa que el 6809, aunque se dice que está provisto de un bus de direcciones de 16 bits y con 64 K de memoria de direccionamiento directo, en realidad sólo dispone de un máximo de 56 K sin hardware ni software de gestión de memoria.

Es posible para algunos dispositivos de E/S el estar conectados al bus de datos del sistema directamente, pero lo común es que haya un chip de interface entre ambos. Estos chips de interface son unos sutilísimos dispositivos tan complejos como el mismo microprocesador, y es corriente el uso de tales chips con un procesador de la misma familia, ya que facilita la tarea de su conexión y control. Los dos chips más comúnmente usados con el 6809 son el PIA 6802 o 6821 (*Peripheral Interface Adaptor*: adaptador de interface de periféricos), que trata las E/S en paralelo, y el ACIA 6850 (*Asynchronous Communications Interface Adaptor*: adaptador asíncrono de interface para comunicaciones), que trata las E/S en serie. Cada una tiene un número determinado de registros, y su control depende de la lectura y escritura de los contenidos almacenados en ellos, pues son considerados como otras tantas posiciones de memoria. Hay tres clases de registros:

- *Registros de control*: Son registros de sólo escritura; se almacenan en ellos valores programados en el chip según las opciones particulares que se desean, como, por ejemplo, el establecer una velocidad en baudios.
- *Registros de estado*: Son registros de sólo lectura, y sus valores indican el "estado" del chip. Por ejemplo, mostrarán si se ha recibido una entrada, o si se transmitió ya la última salida, o si se ha producido un error.



Periférico esencial

Las impresoras necesitan recibir los datos en formatos y velocidad determinadas: no es práctico que la CPU se encargue de tareas tan triviales, y por esta razón lo que hace ésta es enviar los datos de los caracteres al PIA, adaptador de la interface de periféricos, que emplea todo su tiempo en la comunicación con la impresora

Kevin Jones

Registro	Despl.
A de datos	0
A de dir. de datos	0
A de control/estado	1
B de datos	2
B de dir. de datos	2
B de control/estado	3

Registro	Despl.
Reg. de control	0
Reg. de estado	0
Datos transmisión	1
Datos recepción	2

• *Registros de datos*: Son registros que contienen datos de entrada o de salida, o sea, son de lectura-escritura o bien de lectura y escritura por separado. Para ahorrar espacio de memoria varios de estos registros ocupan la misma dirección. Por ejemplo, un registro de estado y otro de control pueden estar en la misma dirección; el que aparecerá en la dirección en cada momento dependerá de lo que se pretenda hacer, si leer o escribir en ella. De forma similar, un registro de datos de entrada y otro de salida pueden compartir la misma dirección.

El PIA 6820 contiene seis registros y ocupa cuatro bytes seguidos del espacio de memoria. El chip en sí está provisto de dos puertas independientes, cada una de las cuales utiliza tres registros. La sección de periféricos del chip tiene ocho líneas de datos y dos líneas de control por cada puerta. Las dos líneas de control se conectan en el periférico a las líneas de control adecuadas de tal modo que puedan servir para determinar el estado. La línea de control 1 sólo trata señales de control de entrada, pero la línea de control 2 puede ser programada para recibir o enviar señales de control.

Los tres registros son:

• Un registro de datos, que puede funcionar como entrada o salida, dado que cada bit puede ser activado independientemente.

• Un registro de dirección de datos, cuyos bits pueden servir individualmente para activar el bit correspondiente a entrada (0) o salida (1) en el registro de datos.

• Un registro mixto de control y estado.

El registro de dirección de datos y el registro de datos comparten la misma dirección. El estado de uno de los bits del registro de control indicará cuál de ellos está presente en un momento dado en la dirección. En la tabla al margen damos el desplazamiento de la dirección de base del chip, para obtener la dirección de cada uno de los registros.

Los bits del registro de control/estado tienen la siguiente asignación:

Bit	Función
7	Bit de estado para la línea de control 1; se pone a uno cuando es recibida la señal de control y pasa automáticamente a cero cuando se lee el registro de datos
6	Bit de estado para la línea de control 2; funciona como el bit 7
5	Determina si la línea de control 2 se emplea para una entrada (0) o para una salida (1)
4	Determina la naturaleza de la señal de control en la línea 2
3	Si la línea de control 2 se establece para entrada, entonces un uno en este bit 3 permite la interrupción del bit 6; si era para salida, sirve para determinar la naturaleza de la señal
2	Selecciona el registro de datos (1) o el registro de dirección de datos (0)
1	Determina la naturaleza de la señal de control en la línea 1
0	Si está a 1 permite la interrupción del bit 7

De momento dejaremos de considerar el empleo de las interrupciones, así como el detalle de los efectos de los bits 1 y 4. Observe que cuando se escribe en el registro para establecer los bits de control es imposible afectar los bits 6 y 7.

El primero de nuestros programas ejemplo crea y emplea un chip 6820 para controlar una impresora a través de una interface Centronics. El segundo especifica un gran número de líneas de control así como las ocho líneas de datos. No vamos a adentrarnos en su detalle, sólo haremos notar que una línea de control (llamada *strobe*) sirve para avisar a la impresora de que hay un carácter en camino. Ésta será conectada con la línea de control 2, la cual será puesta para salida. Otra señal de control (denominada *acknowledge*: reconocimiento) es usada por la impresora para indicar que está dispuesta para recibir el siguiente carácter. Ésta será conectada con la línea de control 1. Las ocho líneas de datos deben, naturalmente, ser conectadas a las ocho salidas de datos de la puerta PIA.

Para activar la puerta hay que seleccionar el registro de dirección de datos y establecer en salida la línea de control 2. Para emplear el chip debemos leer continuamente el registro de control/estado hasta que aparezca un 1 en el bit 7, que nos indicará que la impresora está preparada para recibir el carácter. Podemos a continuación escribir éste en el registro de datos, el cual envía automáticamente una señal de control por la línea de control 2. El bit 6 se pondrá a 1 cuando el carácter haya sido transmitido. Después se leerá el registro de datos para borrar los bits 6 y 7 y repetir el proceso hasta que haya sido transmitido el último carácter. El proceso de envío y recepción de señales de control entre el procesador y el periférico se conoce como *handshaking* (apretón de manos).

Supondremos que la dirección base del PIA se encuentra en una tabla de direcciones situada en $3000. Al entrar en la subrutina de impresión, el registro A del procesador contiene el índice de esta tabla, y el registro Y contiene la dirección de la serie de caracteres a imprimir. Esta cadena está almacenada en el formato normal, es decir, con el byte que indica su longitud en primer lugar. Hay dos subrutinas: una para activar la puerta y otra para imprimir la cadena.

El ACIA 6850 es un UART (transmisor/receptor asíncrono universal) empleado en la comunicación en serie, con el protocolo RS232 generalmente y un modem las más de las veces. Dispone de cuatro registros y ocupa dos direcciones. En el lugar de los periféricos se encuentran cinco conexiones con el chip: una línea para la transmisión de datos, otra para su recepción, y tres líneas de control para el *handshaking*, si así se necesita. Dos de ellas están destinadas a señales de control de entrada —la DCD (*Data Carrier Detect*: detección del portador de datos) y la CTS (*Clear To Send*: puesta a cero para envío)— y la última sirve para las señales de salida: RTS (*Request To Send*: petición de envío). El empleo de tales líneas se deduce de sus denominaciones, y han de conectarse a las líneas denominadas de igual modo dispuestas en el RS232 estándar.

Los cuatro registros del ACIA están escritos al margen de esta página. En el registro de control, el bit más significativo (el 7) sirve para permitir interrupciones en la recepción de datos. Los bits 5 y 6 se emplean para permitir o inhibir interrupciones en la transmisión y para determinar la naturaleza de la señal de control enviada por la línea RTS. Los bits 2, 3 y 4 sirven para determinar el tamaño del "paquete" transmitido en cada momento. Cuando se transmite un byte sobre un enlace en serie por lo general se envían al menos 10 bits, comenzando con el bit de inicio, detectado por el receptor para advertirle que hay datos en camino. Los datos en sí pueden ser siete u ocho bits, pero puede que se



añada un bit de paridad, es decir, un bit extra para la detección de errores en la transmisión. Finalmente, puede que se incluyan uno o dos bits de detección de final. He aquí las diferentes opciones:

Registro de control Bit 4	Bit 3	Bit 2	Número de bits de datos	Paridad	Número de bits de final
0	0	0	7	par	2
0	0	1	7	impar	2
0	1	0	7	par	1
0	1	1	7	impar	1
1	0	0	8	nula	2
1	0	1	8	nula	1
1	1	0	8	par	1
1	1	1	8	impar	1

Los dos bits menos significativos (el bit 0 y el 1) se emplean para determinar la velocidad de transmisión y de recepción. Esto se consigue estableciendo un divisor del paso de reloj. El 6850 no tiene reloj propio, por lo que debemos proporcionarle uno externo, comúnmente a 1 760 Hz.

Registro de control Bit 1	Bit 0	Paso de reloj divisor
0	0	1
0	1	16
1	0	64

En la tabla se omite la cuarta posibilidad (ambos bits a uno) que ocasiona la reinicialización total del chip. En el registro de estado los bits tienen las siguientes funciones:

Bit	Función
7	Petición de interrupción
6	Se pone a 1 si ha ocurrido algún error en la recepción
5	Se pone a 1 si hay una sobreescritura en el receptor, es decir, si los caracteres se sobreponen a los previamente recibidos
4	Se pone a 1 si ha habido un error de enmarque (*framing*) en la recepción, o sea, hay un número equivocado en los bits de inicio o de final
3	Se pone a 1 cuando se recibe una señal por la línea CTS
2	Se pone a 1 cuando se recibe una señal en la línea DCD
1	Se pone a 1 cuando el reg. de transm. de datos está vacío
0	Se pone a 1 cuando el reg. de datos recibidos está lleno

Nuestro programa del segundo ejemplo emplea un chip 6850 para recibir una cadena de caracteres, acabando en un retorno de carro, transmitidos de un terminal remoto. El principio consiste en programar el chip adecuadamente y hacer que un bucle compruebe si el registro de datos recibidos está completo. Cuando esto sucede, retiramos el byte de datos, lo cual reinicializa el bit 0 del registro de estado. Este proceso se repite hasta que el carácter recibido es el correspondiente al retorno de carro (el código 13 en ASCII). Nos despreocuparemos de cualquier posible error de transmisión, aunque no resulta nada difícil la comprobación a través de una máscara para los contenidos del registro de estado que compruebe si se ha puesto a 1 alguno de los bits indicadores de error de transmisión. Asumiremos un protocolo bastante común: 8 bits de datos, paridad nula, 2 bits de final y una velocidad de reloj dividida por 16. La primera subrutina programa el chip, la segunda recibe los datos.

Programa para PIA

```
TABLE   EQU   $3000         Subrutina para establ. Puerta A
        ORG   $1000
        ASLA                Desplaza A a la izq. para
                            multiplicar por 2 (la tabla
                            contiene dirs. de 2 bytes)
        LDX   #TABLE        Obtiene la dir. base del PIA
        LDX   A,X
        CLR   1,X           Obtiene acceso al registro de
                            dirección de datos
        LDB   #%11111111    Pone a 1 los bits para salida
        STB   ,X
        LDB   #%00101100    Inhibe las interrupciones,
                            activa lín. cont. 2 para sal. y
                            selec. el reg. de datos
        STB   1,X
        RTS
                            Subrutina para imprimir la cadena
                            cuya dirección se encuentra en Y
        ASLA                Desp. A a la izq. para mult. por 2
        LDX   #TABLE        Obtiene dir. base del PIA
        LDX   A,X
        LDA   ,Y+           Obtiene long. de la cad. en A
LOOP1   BEQ   FINISH        Comprueba si long. es cero
LOOP2   LDB   1,X           Compr. si está disp. para sig. bit
        ANDB  #%10000000    Enmascara bits excepto 7
        BEQ   LOOP2         Si no está preparada
        LDB   ,Y+           Obtiene el siguiente carácter
        STB   ,X            Lo imprime
LOOP3   LDB   1,X           Comprueba si fue transmitido
        ANDB  #%01000000    Mira el bit 6
        BEQ   LOOP3         Bucle si no está aún prepar.
        LDB   ,X            Lee reg. de datos para poner a
                            0 los bits de estado
        DECA                Resta 1 a la longitud
        BRA   LOOP1         Toma el sig. carácter
FINISH  RTS
```

Programa para ACIA

```
TABLE   EQU   $3000         Subrutina para programar el 6850
        ORG   $1000
                            Subrutina para est. el ACIA
ACIAST  ASLA                Desp. A a la izq. para mult.
                            por 2 (tabla dirs. de 2 bytes)
        LDX   #TABLE        Obtiene dir. base del ACIA
        LDX   A,X
        LDA   #%00000011    Reinicialización total del ACIA
        STA   ,X            Pone en reg. de control
        LDA   #%00010001    Progr. el ACIA (8 bits de datos,
                            no hay paridad, 2 bits de final)
        STA   ,X
        RTS
                            Subrutina para aceptar cadena de
                            caracteres
BUFFER  EQU   $4000         Algún lugar para colocar cad.
CR      EQU   13            Código ASCII retor. de carro
        BSR   ACIAST        Establece el ACIA. El registro
                            X contiene la dir. del ACIA
        LDY   #BUFFER       Destino en Y
LOOP    LDB   ,X            Obtiene el estado
        ASLB                Desp. el bit 7 fuera del reg. B
                            y lo introd. en flag de arras. CRC
        BCC   LOOP          Retrocede si ese bit no se
                            puso a 1, es decir, que no se
                            solicitó aún interr. alguna
        LDA   1,X           Toma el byte de datos
        STA   ,Y+           Lo almacena
        CMPA  #CR           ¿Se trata del retor. de carro?
        BNE   LOOP          Siguiente carácter
        RTS
```

Guerra en la galaxia


Encuentro con los Zylones



Mapa galáctico

Ian McKinnell

"Star raiders", de Atari, es un desarrollo, especial para ordenadores personales, del popular juego "Star trek"

En *Star raiders*, el jugador asume el papel de comandante de la nave espacial Star Raider y viaja a través de la galaxia en busca de las naves de los enemigos, los Zylones. El juego requiere el empleo del teclado además del control por palanca de mando. Después de desplazarse hasta un sector de la galaxia, pulsando la tecla F se visualiza la panorámica frontal desde la cabina. La posición de las naves Zylon se señala mediante indicadores en la parte inferior de la pantalla, y pulsando la tecla L el jugador llama al "explorador de largo alcance", que proporciona una panorámica del sector de la cuadrícula en curso con la nave del jugador en el centro y las Zylon formadas en la distancia.

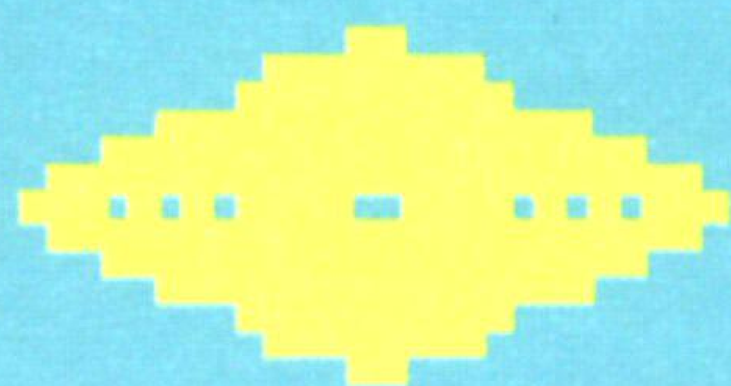
El jugador debe entonces atacar al enemigo, ya sea utilizando los motores normales, lo que bien podría conducir a que las naves Zylon escaparan, o bien adentrándose en el "hiperespacio" (lo que se consigue pulsando H), en cuyo caso la distancia se cubrirá en unos pocos segundos. Antes de lanzarse al hiperespacio, debe utilizarse el ordenador perseguidor pulsando T; si no se hace esto, el salto al hiperespacio bien podría conducir a que la nave del jugador acabara en un sector totalmente desconocido de la galaxia. Otros factores a considerar son la utilización del ordenador de ataque (al que se accede mediante la tecla C) y los escudos defensores del Star Raider, que se encienden mediante el empleo de la tecla S.

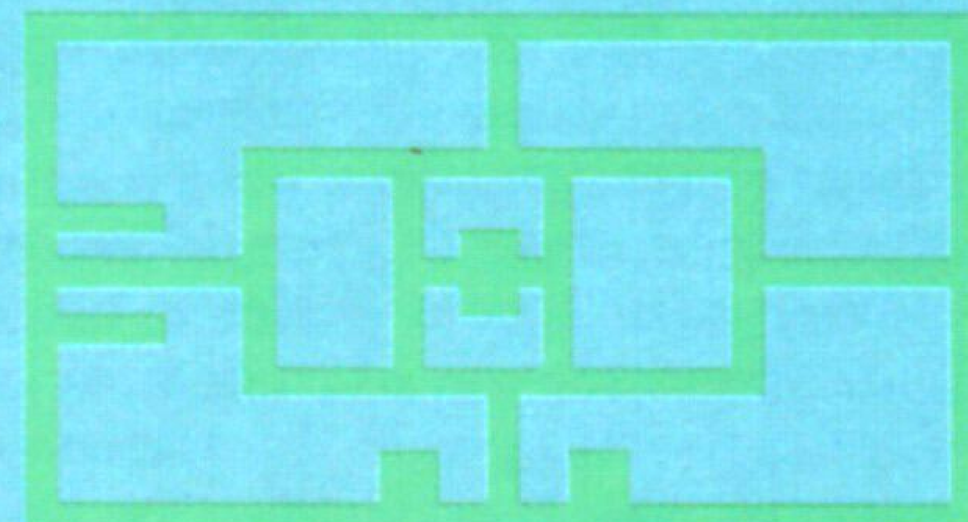
Una vez abandonado el hiperespacio, el ordenador enciende una luz intermitente de "alerta roja" y comienza la batalla. Las naves Zylon atacan la Star Raider desde todos los lados, haciéndose cada vez más grandes a medida que se acercan. El control con la palanca de mando permite al jugador hacer rotar la nave en todas las direcciones, y la velocidad de la misma se determina mediante el empleo de las teclas numéricas. La medida de control ofrecida permite que el jugador intervenga en escaramuzas aéreas, descendiendo en picado y arremetiendo contra el enemigo. Pero éstas consumen muchísima energía y la flota enemiga hará blanco repetidamente en la Star Raider, lo que obligará al usuario a visitar la base estelar para reabastecerse de combustible y efectuar las necesarias reparaciones. Deberá, entonces, trasladarse a una cuadrícula que contiene una estrella; a medida que la Star Raider se acerca, la estrella irá creciendo y se transformará en un enorme platillo volante amarillo. Entonces se produce la maniobra más difícil del juego, con el jugador intentando colocarse en órbita alrededor de la base. En primer lugar, es preciso que la nave esté fija en los visores de la nave, y el jugador debe retardar el impulso de ésta hasta que el indicador de distancia al objetivo marque cero. Una vez sucede esto, se puede detener a la Star Raider y, si la maniobra ha sido correcta, el mensaje ORBIT ESTABLISHED (establecida órbita) aparecerá intermitente en la parte superior de la pantalla. Durante esta aproximación se debe tener un enorme cuidado, puesto que es extremadamente fácil errar el objetivo. Después de establecida la órbita, de la base estelar emerge una nave auxiliar de reaprovisionamiento de combustible y se acopla con la Star Raider, permitiendo que la nave retorne al combate.

De cuando en cuando aparece en la pantalla un mensaje STARBASE SURROUNDED (base estelar rodeada) y el jugador debe, entonces, dirigirse apresuradamente hacia la estación amenazada para impedir su destrucción. Cuando intenta defender la base estelar, el jugador debe dedicar sumo cuidado a evitar que las armas de la Star Raider hagan blanco en la base estelar.

Star raiders ofrece cuatro niveles de juego, que van desde "novato" a "comandante". En los niveles inferiores, el jugador no necesita preocuparse demasiado por el daño que se le pueda infligir a la Star Raider, dado que hay muy pocos Zylones y la puntería de éstos no es muy certera. En los niveles superiores resulta sumamente difícil sobrevivir.

Star raiders: Para todos los ordenadores Atari
Editado y distribuido por: Atari Co., AUDELEC, Compás de la Victoria, 3. 29012 Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Cartucho

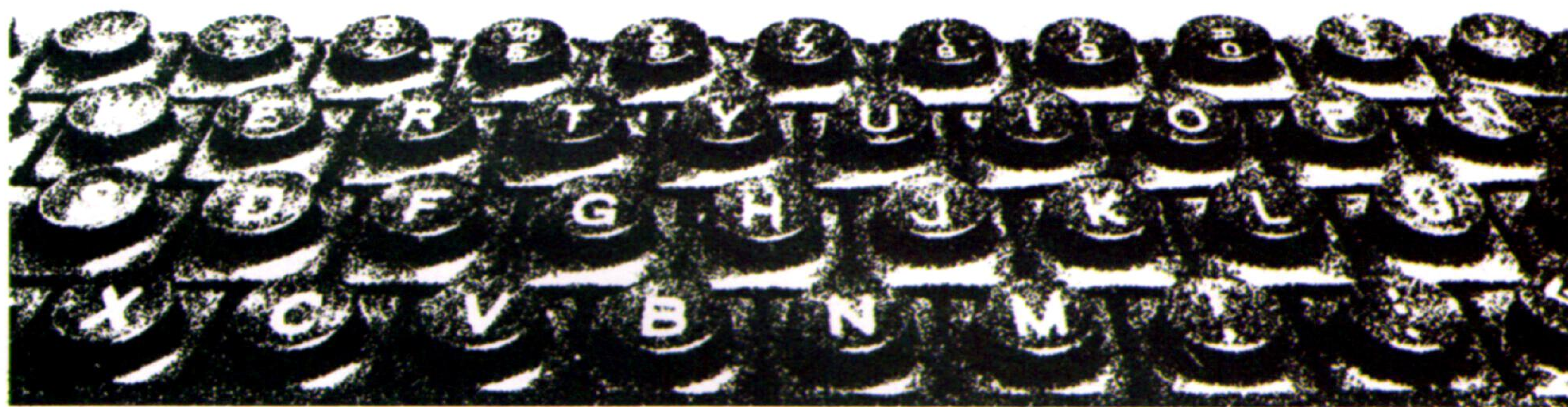




Liz Dixon



mi computer
Editorial Delta, S.A.



E R T Y U I O P
D F G H J K L
X C V B N M


10058
9 788485 822836